Arto Niittukari

Excel-työkirjaan sisäänrakennetun laskenta-työkalun toteutus verkkosovelluksena

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikan koulutusohjelma

Insinöörityö

13.5.2015

Metropolia

Tekijä Otsikko Sivumäärä Aika	Arto Niittukari Excel-työkirjaan sisäänrakennetun laskentatyökalun toteutus verkkosovelluksena 55 sivua + 2 liitettä 13.5.2015
Tutkinto	insinööri (AMK)
Koulutusohjelma	tietotekniikan koulutusohjelma
Suuntautumisvaihtoehto	ohjelmistotekniikka
Ohjaaja	lehtori Pasi Ranne

Insinöörityön tarkoituksena oli kehittää Microsoft Excel -taulukkolaskentaohjelman työkirjatiedostoon sisäänrakennetusta laskentatyökalusta kaupalliseen tarkoitukseen tuleva verkkosovellus. Ajatuksena oli, että sovellus tulisi toteuttamaan SaaS-pilvitoimintamallin toimintaperiaatteen, jossa käyttäjä ostaa ohjelmiston määritetyksi ajaksi käyttöönsä palveluna ilman omistajuuden vaihtoa.

Insinöörityön tilaaja on laskentapalveluiden tarjoamiseen erikoistunut yritys. Tilaaja oli nähnyt alun perin omaan käyttöönsä kehittämässään laskentatyökalussa kaupallista potentiaalia ja halusi tästä syystä kehittää työkalusta kokonaisvaltaisemman version, joka tukisi useita käyttäjiä ja käyttäjätilien hallintaa. Sovelluksella työstetään aina yksittäistä sovelluksen sisäistä projektia, jonka sovelluksen käyttäjä voi tallentaa joko palveluntarjoajan palvelimella sijaitsevaan tietokantaan tai omalle työasemalleen. Varsinaista käyttöarvoa käyttäjät saavat erilaisista sovelluksen koostamista raporteista, joita käyttäjät pystyvät tallentamaan omille työasemilleen CSV- ja PDF-tiedostoina tai katsomaan suoraan selaimella HTML-esitysmuodossa.

Laskentasovelluksen palvelinpuolen kehittämiseen käytettiin PHP-ohjelmointikielellä toteutettua Laravel-sovelluskehystä, jolla saatiin toteutettua sovelluksen runko. Sovellukseen tuli kehittää pysyviä tietoja varten tietokantatallennusta ja väliaikaisia tietoja varten istuntotallennusta. Sovellus toteutettiin yhden sivun verkkosovelluksena käyttäen sovelluksen asiakaspuolen kehittämiseen JavaScript-ohjelmointikielellä toteutettua AngularJS-sovelluskehystä. Sovelluksen käyttöliittymä toteutettiin HTML- ja CSS-tekniikoilla, mutta käyttöliittymän ulkoasu jäi insinöörityön osalta sovitusti hyvin yksinkertaiseksi.

Tavoitteena oli saada aikaiseksi toimiva verkkosovellusprototyyppi, joka sisältäisi kaikki insinöörityön alussa määritetyt ominaisuudet. Tavoite saatiin toteutettua, ja tilaaja oli tyytyväinen lopputulokseen. Sovellus ei kuitenkaan ole vielä julkaisukelpoinen ja siihen on jo suunniteltu useita parannuksia ja lisäominaisuuksia. Sovellus saatiin kuitenkin sellaiseen tilaan, että sitä voidaan alkaa jatkokehittää.

Avainsanat	laskentatyökalu, ohjelmisto palveluna, Laravel, AngularJS, yhden sivun sovellus, verkkosovellus

Author Title Number of Pages Date	Arto Niittukari Implementing a Web Application Based on a Computation Tool Built into an Excel Workbook 55 pages + 2 appendices 13 May 2015
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor	Pasi Ranne, Senior Lecturer

The objective of this thesis was to develop a web application based on a computation tool built using Microsoft Excel. The application was meant to be developed for commercial purposes making it fall in the SaaS category of the cloud computing stack.

The thesis was made for a company specialized in computation services. The application was designed to be project-based by having the ability to save projects to the workstation of the user and the database of the service provider. The application generates downloadable CSV and PDF reports and HTML reports which can be read directly from the browser.

The Laravel PHP framework was used to develop the server-side functionality of the application, including database and session storage implementations. The application was developed as a single-page application using the AngularJS JavaScript framework in the client-side development of the application. HTML and CSS were used to develop the user interface of the application but the appereance was left very simple-looking.

A working web application prototype got produced and the company was satisfied with the outcome. However, several improvements and additional features have already been introduced for the application. Nonetheless, the application got developed to a reasonable state where it can be further developed.

Keywords	computation tool, Software as a Service, Laravel, AngularJS, single-page application, web application

Sisällys

Lyhenteet ja käsitteet

Ajax	Joukko verkkosovellustekniikoita, joiden avulla sovelluspalvelimeen voidaan olla yhteydessä asynkronisesti. Lyhennelmä sanoista Asynchronous JavaScript + XML, mutta käytetään erisnimenä.
AngularJS	JavaScript-ohjelmointikielellä toteutettu avoimeen lähdekoodiin perustuva SPA-arkkitehtuuria noudattava verkkosovelluskehys verkkosovellusten asiakaspuolten ohjelmointiin.
API	Application Programming Interface. Ohjelmointirajapinta, joka määrittää ohjelman toiminnan ja ohjeet sen käyttöön.
Asiakaspuoli	Verkkosovelluksissa asiakasohjelmalle eli verkkoselaimelle määritetty toiminnallisuus.
CSS	Cascading Style Sheets. Tyyliohjeiden laji, jonka avulla saadaan määritettyä, miten HTML-elementit näytetään.
CSV	Comma-Separated Values. Tiedostoformaatti, joka mahdollistaa taulukkomaisten tietojen esittämisen ja yksittäisten tietojen erottamisen toisistaan pilkun tai jonkin muun erottimen avulla.
DOM	Document Object Model. Malli ja ohjelmointirajapinta mm. HTML-dokumenttien sisältöjen esittämiseen ja muokkaamiseen.
HTML	Hypertext Markup Language. Merkkauskieli, jonka avulla verkkoselaimelle määritetään, miten verkkosivujen sisällöt on tarkoitus näyttää.

HTTP	Hypertext Transfer Protocol. Protokolla, joka määrittää säännöt asiakasohjelman ja verkkosivuista vastaavan palvelimen vuorovaikutukselle.
JavaScript	Verkkosivujen- ja sovellusten asiakaspuolten ohjelmointiin käytetty ohjelmointikieli.
JSON	JavaScript Object Notation. Kevytrakenteinen formaatti tiedon esittämiseen ja siirtämiseen.
Käsittelijä	Controller. MVC-arkkitehtuurin osa, joka ohjaa ohjelman suoritusta mallin ja näkymän välillä.
Käänteisen hallinnan säiliö	IoC (Inversion of Control) container. Työkalu, jonka avulla ohjelmoinnissa tarvittavia luokkia saadaan käyttöön riippuvuusinjektiona.
Laravel	PHP-ohjelmointikielellä toteutettu avoimeen lähdekoodiin perustuva MVC-arkkitehtuuria noudattava verkkosovelluskehys verkkosovellusten palvelinpuolten ohjelmointiin.
Malli	Model. MVC-arkkitehtuurin osa, joka kuvaa sovellukseen liittyviä loogisia toimintoja ja tietoja.
MVC	Model-View-Controller. Ohjelmistoarkkitehtuuri, jonka tarkoituksena on sovelluksen käyttöliittymän erottaminen sovelluslogiikasta.
Näkymä	View. MVC-arkkitehtuurin osa, jonka tarkoituksena on kuvata sovelluksen visuaalisia esitysmuotoja.
Palvelinpuoli	Verkkosovelluksissa sovelluksesta vastaavalle palvelimelle määritetty toiminnallisuus.

PDF	Portable Document Format. Tiedostoformaatti, jonka sisältö pystytään tulostamaan paperille pääosin samannäköisenä, kuin miltä se näyttäisi ohjelmistolla avattuna.
PHP	Hypertext Preprocessor. Suosittu verkkosovellusten palvelinpuolten ohjelmointiin käytetty ohjelmointikieli.
Pilvipalvelu	Palvelu, jonka periaatteena on tarjota verkon välityksellä tietotekniikkaresursseja asiakkaan käyttöön ilman omistajuuden vaihtoa.
Riippuvuusinjektio	Dependency Injection. Menetelmä, joka luo sovelluskehittäjälle joustavuutta ohjelmoinnissa tarvittavien riippuvuusluokkien käyttöön ja vaihtamiseen.
SaaS	Software as a Service. Pilvitoimintamalli, jossa asiakkaalle tarjotaan palveluna jokin ohjelmisto, jota käytetään tyypillisesti verkkoselaimella internetyhteyden välityksellä.
Sovelluskehys	Sovelluskehityksen avuksi tuotettu joukko valmiita kirjastoja ja paketteja, jotka helpottavat sovelluskehitysprosessia sovellukselle tavanomaisten toimenpiteiden kanssa.
SPA	Single-Page Application. Yhden HTML-sivun varaan rakennettu verkkosovellus, joka poistaa kokonaiset sivunlataukset sovelluksen elinkaaresta.
Säilömalli	Repository Pattern. Tietojen tallennukseen kehitetty suunnittelumalli, joka erottaa tallennusmenetelmän toteutuksen MVC-arkkitehtuurin käsittelijästä.
Virtualisointi	Pilvipalveluiden toteuttamiseen käytetty tekniikka, joka mahdollistaa kapasiteetti- ja ohjelmistopalveluiden tuottamisen ilman fyysisiä laitteita.

1 Johdanto

Internet on koko ajan ympärillämme. Siitä on tullut merkittävä osa kuluttajien ja yritysten arkea. Internetiin liittymisen kustannus on nykyään niin alhainen, että siihen on mahdollista liittää mitä tahansa laitteita. Yritys- ja kotitietotekniikan yrityksissä on suurta kaupallista kiinnostusta hyödyntää internetiä sovellusten tarjoamisen kanavana. Monia pieniäkin tuotteita on liitetty ja tullaan jatkossa liittämään internetiin kysynnän puitteissa. Tämä voi pitää sisällään esimerkiksi ihmisten viestintätarpeiden, kiinnostusten kohteiden ja osaamisen mukaisia sovelluksia. Pääasiallisesti nämä palvelut saadaan myös halvemmalla kustannuksella kuin esimerkiksi perinteisemmät työpöytäsovellukset. Tekninen mahdollisuus ja toteuttamisen helppous eivät kuitenkaan vielä takaa kannattavuutta, eikä ilman kannattavuutta synny laajamittaista tarjontaa. Sovelluksia ei siis välttämättä ole tarjolla erikoisempiin käyttötarkoituksiin. [1, s. 9–11.]

Insinöörityön perustana on laskentatyökalu, joka on sisäänrakennettu Microsoft Excel -taulukkolaskentaohjelman työkirjatiedostoon. Työkalun laskentalogiikka on toteutettu Excelin valmisfunktioilla ja muu toiminnallisuus Excelin tukemilla automatisoiduilla komentosarjoilla eli makroilla. Työkaluun on jopa toteutettu yksinkertainen käyttöliittymä, joka sisältää navigointia ja tiedonsyöttölomakkeita. Alun perin työkalu on kehitetty insinöörityön tilaajan omaan käyttötarkoitukseen ja se on toiminut laskentaa suorittavana apuvälineenä. Tilaaja kuitenkin näki työkalussa kaupallista potentiaalia ja päätti kehittää siitä sovelluksen kaupalliseen tarkoitukseen. Tämän työkaluprototyypin pohjalta olisi tarkoitus kehittää sovellus, joka noudattaisi samaa taulukkomaista laskentalogiikkaa, mutta olisi kuitenkin sovelluksena kokonaisvaltaisempi. Valmiiseen sovellukseen tulisi lopulta mahdollisuus ostaa lisenssi, joka oikeuttaisi sovelluksen käytön määritetyn kauden aikana.

Tilaajan alkuperäisenä toiveena oli, että insinöörityönä kehitettävä sovellus toteutettaisiin käyttäjän omalle työasemalle asennettavana verkkosovelluksena, jolloin sovellusta käytettäisiin verkkoselaimella ja sitä olisi mahdollista käyttää ilman internetyhteyttä. Perusteena tälle ratkaisutavalle oli se, että sovellusta voitaisiin aluksi myydä pienemmille asiakasmäärille asennettavana versiona, mutta kysynnän mahdollisesti kasvaessa voitaisiin sovellus tarjota asiakkaille palveluna internetin välityksellä. Toivottuun toteutustapaan sisältyy kuitenkin useita ongelmia, joiden vuoksi vastaavanlaista sovelluksen väliaikaisversiota ei ole kannattavaa toteuttaa.

Verkkosovelluksen sijaitseminen käyttäjän työasemalla tarkoittaa sitä, että käyttäjällä on koko ajan pääsy kaikkiin sovelluksen lähdekoodeihin. Käyttäjän olisi siis mahdollista muokata lähdekoodeja tai vaikka kopioida ne kokonaan itselleen. Työn ehdottomana vaatimuksena on, että käyttäjällä ei tulisi olla varsinkaan sovelluksen laskentalogiikan lähdekoodeihin minkäänlaista pääsyä. Läpäisemättömän lisenssitarkistuksen toteuttaminenkin olisi paikallisen asennuksen vuoksi käytännössä mahdotonta. Lähdekoodien salaamiseen on olemassa käytetyistä teknologioista riippuen eri menetelmiä, mutta salaukset voidaan silti purkaa. Lähdekoodien lukemista saadaan toki vaikeutettua salausmenetelmillä, mutta alkuperäinen ongelma ei varsinaisesti häviä mihinkään. [2.]

Myös tallennusmenetelmien toteutuksissa tulisi omalle työasemalle asennettavan verkkosovelluksen kanssa vaikeuksia, koska lopulta internetiin siirrettävä sovellus tulee joka tapauksessa tarvitsemaan tietokantapalvelimen ja toteutukset käyttäjäkohtaisten tietojen tallennusta varten. Lisäksi tarvittavien palvelimien asentaminen ja niiden käynnistäminen joka kerta yksittäisen paikallisesti toimivan verkkosovelluksen käytön yhteydessä olisi hankalaa ja tarpeettoman raskasta.

Näiden ongelmakohtien tiedostamisen jälkeen tulimme tilaajan kanssa yksimielisesti siihen lopputulokseen, että ei ole kannattavaa toteuttaa väliaikaista versiota sovelluksesta. Erilaiset yritysten verkkosivujen- ja sovellusten ylläpitoon ja julkaisuun tarkoitetut webhotellipalvelut koettiin sen verran edullisiksi, että resurssien kuluttaminen väliaikaisen sovellusversion kehittämiseen kustantaisi joka tapauksessa enemmän kuin sovelluksen menestymisen kokeileminen suoraan internetissä.

Toive sovelluksen käytöstä ilman internetyhteyttä jää päätöksen vuoksi kuitenkin toteuttamatta. Tämä ei kuitenkaan ole erityisen suuri ongelma, sillä liikkuvat laajakaistayhteydet mahdollistavat päätelaitteiden liittämisen internetiin lähes missä vain ja yhteydet Suomessa tulevat mahdollisesti vain paranemaan lähivuosien aikana. [3.]

Excelillä rakennettu laskentatyökalu sisältää jo itsessään hyvin paljon toimintalogiikkaa, mutta verkkosovellusta varten tulisi toteuttaa myös muutamia lisäominaisuuksia. Sovelluksen olisi ensinnäkin tarkoitus toimia projektipohjaisesti. Tämä tarkoittaa sitä, että sovellusta käytettäessä työstetään aina jotakin projektia ja projekteja olisi mahdollisuus tallentaa ja avata sovelluksessa uudestaan. Käyttäjien tulisi pystyä tallentamaan projekteja palveluntarjoajan palvelimella sijaitsevaan tietokantaan tai omille työasemilleen tiedostoina.

Varsinaista käyttöarvoa sovellus toisi käyttäjilleen koostamalla erilaisia raportteja syötettyjen laskentatietojen perusteella. Käyttäjien tulisi pystyä lukemaan raportteja suoraan selaimen näkymästä ja tallentamaan omille työasemilleen kahdessa eri tiedostomuodossa. Toteutukseen soveltuvien teknologiaratkaisujen valitseminen, ominaisuuksien toteuttaminen ja taulukkoajattelusta siirtyminen ohjelmointimaailmaan tulevat todennäköisesti muodostamaan suurimmat haasteet koko projektiin.

Insinöörityötä päätettiin rajata suuren työmäärän vuoksi niin, ettei sovelluksen käyttöliittymän ulkoasua pidettäisi insinöörityön lopputuloksen osalta niin oleellisena. Pääasia on, että alkuperäisen sovellusprototyypin koko toimintalogiikka saataisiin toimimaan kehitettävässä verkkosovelluksessa määritetyt lisäominaisuudet mukaan lukien.

2 Ohjelmisto palveluna

2.1 Pilvipalvelut

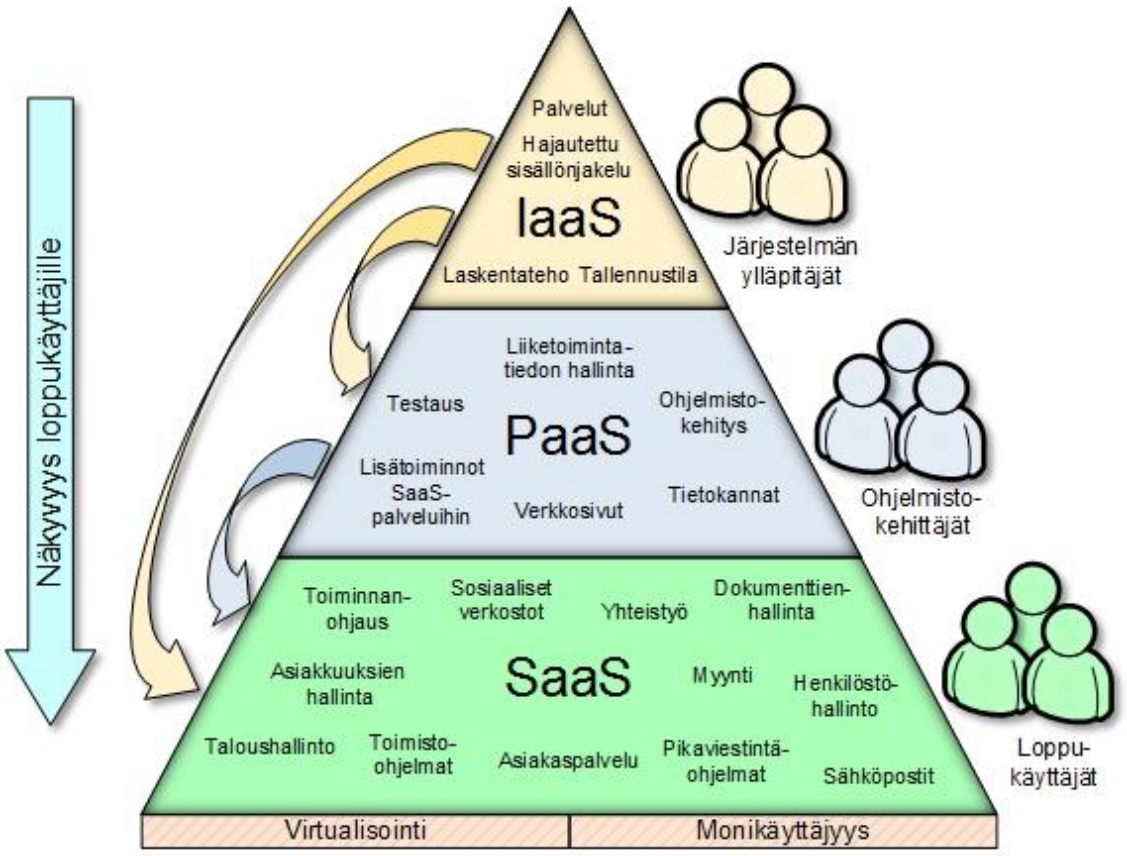
Internetin ja tietotekniikan alan kehitykselle ominainen piirre on jatkuva innovointi ja sen myötä uusien käsitteiden ja toimintamallien esittely. Pilvilaskenta on käsite, joka on viime vuosien aikana mullistanut tietotekniikan alaa. Sillä tarkoitetaan tietoteknisten resurssien hajauttamista eri päätelaitteiden muodostamaan kokonaisuuteen eli pilveen, joka on saavutettavissa verkkoyhteyden avulla. Näitä resursseja hyödyntämällä saadaan luotua palveluita, joita kutsutaan pilvipalveluiksi. [4, s. 10.]

Pilvipalveluiden periaatteena on tarjota verkon välityksellä tietotekniikkaresursseja asiakkaan käyttöön ilman omistajuuden vaihtoa ja ilman, että asiakkaan tarvitsee tietää, missä resurssit sijaitsevat. Tietotekniikkaresursseilla voidaan tarkoittaa esimerkiksi tietoliikenneyhteyksiä, laskenta- ja tallennuskapasiteettia tai ohjelmistoja. [4, s. 16.] Näin tuoteajattelusta siirrytään palveluajatteluun ja tietotekniikkapalveluita voidaan saada suoraan jakeluverkosta samalla tavalla kuin esimerkiksi sähköä. Jakeluverkkoa pilvipalvelumaailmassa edustaa tyypillisesti internet, mutta se ei ole pilvipalveluiden välttämätön edellytys. [4, s. 11.]

Virtualisointi on tekniikka, jolla on ollut suuri merkitys pilvitoimintamallin mahdollistumisen kannalta. Virtualisointi mahdollistaa kapasiteetti- ja ohjelmistopalveluiden tuottamisen ilman fyysisiä laitteita, jolloin toimijan tuotantokustannukset alenevat merkittävästi.

Jos virtualisointiin yhdistetään monikäyttäjyys eli ympäristön jakaminen useiden käyttäjien kesken, sitoutuu yhtä tuotettua virtuaalista palvelinlaitetta kohden vielä vähemmän pääomaa. Palveluiden käynnistys- ja purkumaksut saadaan näin ollen hyvin pieniksi, jolloin kapasiteetin tarjoaminen päiväksi, tunniksi tai puhtaasti käytön mukaan tulee mahdolliseksi. [1, s. 59.]

Pilvipalvelut jaotellaan tyypillisesti pilvitoimintamalleihin niiden ominaispiirteiden mukaan. Yleisimmät pilvitoimintamallit ovat IaaS (Infrastructure as a Service), PaaS (Platform as a Service) ja SaaS (Software as a Service). IaaS-mallissa asiakas ostaa palveluntarjoajalta laitteiston resursseja käyttöönsä palveluna. PaaS tarjoaa asiakkaalle valmiin alustan, joka on usein tarkoitettu ohjelmistokehitykseen. SaaS-tyyppiset pilvipalvelut on tarkoitettu erityisesti loppukäyttäjille (kuva 1). Niissä asiakkaalle tarjotaan jokin ohjelmisto palveluna, jota käytetään tyypillisesti verkkoselaimella internetyhteyden välityksellä. [4, s. 20–26.]



Kuva 1. Yleisimmät pilvitoimintamallit, niiden kohderyhmät, esimerkkejä käyttökohteista ja mallikohtainen näkyvyys loppukäyttäjille [4, s. 22–25; 5].

SaaS-palvelut toimivat niin, että omistamisen, asentamisen, ylläpidon ja päivittämisen sijaan asiakas ostaa ohjelmiston käyttöönsä määritetyksi ajaksi ja asiakas vapautetaan ohjelmistoon liittyvän ylläpidon ja päivityksen tuskasta. SaaS-palveluita löytyy moniin tarpeisiin sekä yrityksille että yksityiskäyttäjille. Yritykset käyttävät SaaS-palveluita mm. taloushallinnossa, henkilöstöhallinnossa ja toiminnanohjauksessa. Yksityiskäyttäjälle yleisempiä käyttökohteita ovat sähköpostit, pikaviestintäohjelmat ja toimisto-ohjelmat. SaaS-markkinat ovat liikevaihdollisesti ylivoimaisesti suurimmat, pisimmälle kehittyneet ja niille ennustetaan myös suurinta absoluuttista kasvua. [4, s. 20–26; 6.]

Vaatimustaso palveluna hankittavalle ohjelmistolle on sitä korkeampi, mitä liiketoimintakriittisempi rooli sillä on. Kriittisimmät ohjelmistot ja arkaluonteisimmat tiedot saattavat olla sellaisia, etteivät pilvipalvelut niiden kohdalla tule edes kysymykseen. Lisäksi yrityksillä saattaa olla pienen käyttäjäkunnan käyttöön tarkoitettuja itse tuotettuja tai hankittuja ohjelmistoja, joille ei löydy vastinetta SaaS-maailmasta, eikä sellaista käyttötarkoituksen erikoisuuden takia ole odotettavissakaan. [4, s. 26.]

Seuraavissa alaluvuissa syvennytään tarkemmin insinöörityön kannalta oleellisimpaan SaaS-pilvitoimintamalliin, vaikka monet sille tunnusomaiset piirteet pätevät muihinkin pilvitoimintamalleihin.

2.2 Hyödyt ja mahdollisuudet

Pilvipalvelumarkkinat ovat käytännössä vasta alkutekijöissään, mutta yhä useammat yritykset ovat jo alkaneet hyödyntämään palveluina ostettavia ohjelmistoja yrityskäytössä. Kuluttajat käyttävät SaaS-palveluita jo massoittain käyttämällä pikaviestimiä, sosiaalisia verkkoyhteisöjä ja vertaisverkkopohjaisia tiedostonjako-ohjelmia. SaaS tarjoaakin lukuisan määrän eri hyötyjä ja mahdollisuuksia sekä yrityksille että yksityiskäyttäjille. [4, s. 34.]

SaaS-palvelut ovat kustannuksiltaan halvempia kuin perinteisemmät työpöytäohjelmistot ja tähän löytyy useita syitä. Virtualisointi ja sen mahdollistama jaettu käyttäjäympäristö takaavat sen, että palvelun ylläpitämisen, kohentamisen ja lopettamisen kustannukset voidaan tasata kaikkien asiakkaiden kesken ja pitää täten mahdollisimman alhaisina. Samalla vastuu näistä edellä mainituista toimenpiteistä siirtyy asiakkaalta palveluntarjoajalle. [1, s. 42.]

Asiakkaat säästävät laitteistokustannuksissa, koska ohjelmistojen suoritukset tapahtuvat pääasiassa pilvessä ja ohjelmistojen käyttöön tarvitaan tyypillisesti vain verkkoselainta. Päätelaitteilta ei siis vaadita suurta suorituskykyä. Päätelaiteriippumattomuus takaa myös tuen entistä useammille laitteille, sillä ohjelmistoja ei enää tarvitse sovittaa eri laitteistojen määrittämille vaatimuksille. [1, s. 45–46.]

Yritysasiakkaat eivät tarvitse omaa henkilökuntaansa ohjelmistojen asentamiseen ja ylläpitämiseen, joten näin säästetään myös henkilöstökustannuksissa ja ajassa. Palveluntarjoaja on lisäksi vastuussa ohjelmiston päivittämisestä eikä siitä koidu ylimääräisiä kustannuksia käyttäjille. Ohjelmistopäivitykset eivät myöskään yleensä aiheuta käyttäjilleen häiriöitä, koska päivitykset ovat automatisoituja ja näkyvät välittömästi. [7, s. 25.]

Laadukkaisiin SaaS-palveluihin on rakennettu ominaisuutena elastinen provisiointi, joka tarkoittaa itsepalvelullisuutta. Itsepalvelullisuus mahdollistaa sen, että uusien asiakkaiden mukaan liittäminen, uusien palveluiden tarjoaminen olemassa oleville asiakkaille ja palveluiden käytön lopettaminen on joustavaa tai jopa automaattista eikä asiakkaiden tarvitse välttämättä olla yhteydessä palveluntarjoajan myyntiedustajaan tai asiakaspalvelijaan. Itsepalvelullisuuteen sisältyy usein mahdollisuus maksaa palvelusta luottokortilla. Itsepalvelullisuuskin on pääosin mahdollista virtualisoinnin ansiosta; palvelun pystyttämistä varten ei tarvitse hankkia fyysisiä laitteita eikä sopimuksen purkamisesta veloiteta mitään. Liittymisen ja purkamisen helppouden ansiosta sopimusaika voi myös olla tavanomaista käyttöpalvelua paljon lyhyempi, joten itsepalvelullisuus mahdollistaa myös monet kuormituspiikkien hallinnan mekanismit. Palvelusta voidaan ottaa tarpeen mukaan lisää kapasiteettia ja hylätä lisäkapasiteetti, kun sitä ei enää tarvita. [1, s. 40; 4, s. 17.]

SaaS-palvelut ovat laitteesta, käyttöjärjestelmästä ja käyttöpaikasta riippumattomia ja vaativat ohjelmistojen asentamisen sijaan vain verkkoyhteyden. Näin ollen palvelut kulkevat aina mukana eli ovat käyttäjän sijainnista riippumattomia. Asiakkaiden data sijaitsee pilvessä, joten dokumenttien ja tietojen jakaminen on helppoa ja niiden mukana kulkemisesta ei tarvitse huolehtia. Datan säilytys ja varmuuskopiointi ovat myös palveluntarjoajan vastuulla. Lisäksi ohjelmistot suunnitellaan tyypillisesti niin, ettei ohjelmistokohtaisten dokumenttien ja ohjelmiston versioiden kanssa tulisi ongelmia. [7, s. 24–28.]

2.3 Haasteet ja riskit

SaaS-palveluiden käyttöönotto on etenkin itsepalvelullisuusominaisuuksia sisältävien palveluiden kohdalla nopeaa ja käyttö usein suhteellisen helppoa. Käyttäjiltä piilotetaan suurin osa palvelun toteutukseen vaaditusta infrastruktuurista ja siihen liittyvästä monimutkaisuudesta. Tästä seuraava huolettomuus on suuri houkutus, mutta vaikka palvelun toteutukseen liittyvät monimutkaisuudet ovat poissa silmistä, eivät ne ole silti hävinneet minnekään. [4, s. 36.]

Dataan ja sen säilyttämiseen liittyy olennaisia tietoturva- ja yksityisyyshaasteita. Asiakkaat haluavat tietojensa olevan varmassa tallessa ilman ulkopuolisten tahojen tai edes palveluntarjoajan oman henkilöstön pääsyä tietoihin. Lisäksi asiakkailla voi olla lainsäädännöllisistä tai muista syistä vaatimuksia tiedon säilyttämisen tapoihin ja sijaintiin liittyen. Vaatimukset voivat liittyä esimerkiksi henkilötietojen tai arkaluonteisten tuotekehitykseen liittyvien dokumenttien säilyttämiseen, jota ei välttämättä haluta toteuttaa etenkään ilman salausta missään organisaation ulkopuolella. [4, s. 37.]

Tietoturvahaasteisiin liittyvät omalta osaltaan myös käyttäjien huolimattomuus, pahantahtoisuus ja tietämättömyys. Sosiaaliseksi hakkeroinniksi sanotaan hyökkäyksiä, jotka pyrkivät pääsemään muuten turvalliseen järjestelmään käsiksi sen parissa työskentelevien ihmisten kautta esimerkiksi sähköposteilla tutunoloisista osoitteista tai houkuttelevilla sähköpostien liitetiedostoilla. Käyttäjät saattavat myös käyttää helposti arvattavia salasanoja, samaa salasanaa useassa palvelussa tai yhdistää nämä kaksi huolimattomuustekijää, jolloin murtautumisriski vain suurenee. [4, s. 40.]

Tietoturvan lisäksi dataan liittyvät saavutettavuus- ja pysyvyyshaasteita. Jos pilveen ei saada yhteyttä esimerkiksi käyttökatkoksen aikana, ei päästä myöskään käsiksi sinne tallennettuihin tietoihin. Palvelu, joka toteutetaan kaukana käyttäjästä, on luonnollisesti riippuvainen tietoliikenneyhteyksistä ja palvelun yleisestä toimintavarmuudesta. Datan saavutettavuudenkin suhteen on siis hyvä miettiä datan arvoa ja kuinka suurella varmuudella sen on oltava käytettävissä. [4, s. 39.]

Palveluntarjoajan ja asiakkaiden väliseen luottamukseen liittyy myös useita haasteita. Palvelun käyttäjillä on käytössään vain ne rajapinnat ja työvälineet, joita palveluntarjoaja antaa käyttöön tai tukee. Asiakkailla ei ole pääsyä palveluntarjoajan fyysisiin tiloihin ja asiakkaat eivät usein edes tiedä, missä palvelua tuottava laitteisto sijaitsee. Palve-

luntarjoajat pyrkivät hälventämään näitä huolia sertifikaatein, laatustandardein, markkinointiviestinnällä, maineenhallinnalla ja sopimuksilla. Pääasiassa asiakkaiden on kuitenkin vain luotettava, että palveluntarjoajan tekninen toteutus toimii, kuten pitää. Suorituskyvyn mittaroinnilla voidaan havaita joitakin puutteita ja taloudellisiin menetyksiin voidaan saada palvelutasosopimuksiin perustuen pientä hyvitystä. Mittarointi ja sopimukset auttavat kuitenkin vasta, kun vahinko on jo tapahtunut. [4, s. 44–45.]

Vaikka luottamuksesta puhutaankin, ei se usein ole mahdollista ilman muodollisia sopimuksia. Sopimukset ovat kehittyneet ja niihin kirjataan yhä täsmällisemmin määriteltyjä palvelutaso- ja sisältölupauksia. Löyhät sopimukset voivat puolestaan johtaa varsinkin yrityskäytössä sitä suurempiin ongelmiin, mitä kriittisemmästä toiminnosta on kysymys. Myös lait, asetukset ja erilaiset toimialakohtaiset suositukset ja totutut toimintatavat on huomioitava. Varsinkin henkilötietojen säilyttämistä säädellään tarkoin ja siihen liittyy myös suurta epävarmuutta. [4, s. 44–45.]

Perinteisimpiin työasemalle asennettaviin työpöytäohjelmistoihin verrattuna SaaS-palveluilla on valtava määrä etuja, mutta kaikkeen SaaS ei kuitenkaan ole ratkaisu. SaaS-palveluita ei voida käyttää ilman verkkoyhteyttä, joten työpöytäohjelmisto voi olla parempi jos ohjelmistoa täytyy pystyä käyttämään myös verkkoon yhteydettömässä tilassa. Hidas verkkoyhteys ja palvelun ruuhkaantuminen aiheuttavat myös epämukavuutta palvelun käytössä. Työpöytäohjelmistoissa on usein myös enemmän kustomointimahdollisuuksia ja joitakin erikoisominaisuuksia ei edes pystytä toteuttamaan verkon ylitse esimerkiksi palvelun monimutkaisuuden tai korkeiden laitteistovaatimusten takia. [7, s. 28–29.]

Mikään inhimillinen järjestelmä ei kuitenkaan käytännössä ole eikä tule olemaan täysin toimintavarma. Virheetöntä vaihtoehtoa ei ole tarjolla ja täydelliseen riskittömyyteen ei liiketoiminnassa todennäköisesti koskaan päästä. [4, s. 41–42.]

2.4 Verkkosovellukset

SaaS-palvelut ovat pääasiassa verkkosovelluksia, joten niiden kehitystyössä vaaditaan tietämystä verkkosovellusten kehityksessä käytetyistä teknologioista ja menetelmistä. Verkkosovellukset ovat dynaamisia verkkosivuja, jotka toimivat käyttäjiensä kanssa interaktiivisesti pelkän staattisen datan näyttämisen sijaan [8]. Verkkosovellukset kuu-

luvat yleensä osaksi WWW:tä (World Wide Web) eli maailmanlaajuista hypertekstidokumenttien muodostamaa tietosysteemiä, joka on saavutettavissa yhteydellä maailmanlaajuiseen tietoverkkoon eli internetiin [9]. Internetin perustana toimii HTTP-protokolla (Hypertext Transfer Protocol), joka toimii pyyntö-vastaus-periaatteella asiakasohjelman eli yleensä verkkoselaimen ja resursseista vastaavan palvelimen välillä. Palvelimelta voidaan pyytää dataa HTTP-pyynnön muodossa, jolloin palvelimelta saadaan vastauksena pyydetyt resurssit ja muut tiedot. [10.]

Verkkosovellusten ohjelmointi ja toiminnallisuus voidaan jakaa kahteen kategoriaan: palvelinpuoleen ja asiakaspuoleen. Palvelinpuolen ohjelmoinnilla tarkoitetaan verkkosovelluksesta vastaavalla palvelimella suoritettavan koodin tuottamista. Palvelinpuolen ohjelmoinnin avulla saadaan rakennettua toiminnallisuutta, kuten käyttäjien vuorovaikutusta ja yhteyden muodostamista palvelimen tietokantaan. Sovelluksen käyttäjillä ei ole pääsyä palvelinpuolen lähdekoodeihin. [8.] PHP (Hypertext Preprocessor) on ylivoimaisesti suosituin palvelinpuolen ohjelmointiin käytetty ohjelmointikieli. Sitä käytetään usein LAMP-paketin yhteydessä, jonka lyhenne tulee useimmiten käytettäessä Linux-käyttöjärjestelmästä, Apache-palvelimesta, MySQL-relaatiotietokannasta ja PHP-ohjelmointikielestä. [11; 12; 13.]

Asiakaspuolen ohjelmoinnilla saadaan rakennettua sovelluksen käyttöliittymän toiminnallisuutta ja ulkoasua. Koodit suoritetaan selaimella ja tämän takia koodit ovat myös käyttäjien luettavissa selaimen avulla. Olennaisimmat ohjelmointikielet ja teknologiat asiaspuolen ohjelmoinnissa ovat HTML (Hypertext Markup Language), CSS (Cascading Style Sheets), JavaScript ja Ajax. [8.] HTML on merkkauskieli, jonka avulla selaimelle määritetään, miten sivujen sisältö näytetään. HTML perustuu merkkaustageihin, jotka edustavat HTML-dokumenttien yksittäisiä sisältökohtia. HTML:n uusin standardi on HTML5. [14.] CSS:n avulla määritetään, miten HTML-elementit näytetään eli sen avulla sivuille voidaan lisätä tyylittelyä. CSS3 on uusin CSS-standardi, jossa toiminnallisuudet on jaettu omiin moduuleihinsa. [15.]

Asiakaspuolen toiminnallisuutta voidaan ohjelmoida JavaScript-ohjelmointikielellä, joka perustuu ECMAScript-standardiin. JavaScriptin avulla sivuja näyttävälle selaimelle voidaan antaa käskyjä, joilla tehdään muutoksia esillä olevaan dokumenttiin DOM-ohjelmointirajapinnan avulla (Document Object Model). [16; 17.] Ajaxilla tarkoitetaan joukkoa verkkosovellustekniikoita, joita hyväksikäyttäen sovelluspalvelimeen voidaan olla yhteydessä asynkronisesti. Ajax-tekniikoiden avulla palvelimelle voidaan lähettää

HTTP-pyyntö niin, että pyyntö suoritetaan käyttäjän näkökulmasta taustatapahtumana ilman sivun uudelleenlatausta. Ajax-pyynnöillä voidaan olla yhteydessä esimerkiksi palvelimella sijaitsevaan tietokantaan. [18.]

Verkkosovellusten kehitykseen käytetään usein valmiita kirjastoja ja paketteja sisältäviä sovelluskehyksiä, jotka helpottavat sovelluskehitysprosessia verkkosovelluksille tavanomaisten toimenpiteiden kanssa. Verkkosovelluksille tavanomaisia toimenpiteitä ovat esimerkiksi tietokantayhteyksien luominen ja istunnonhallinta. Sovelluskehykset auttavat myös sovellusrakenteen organisoinnissa, edistävät koodin uudelleenkäyttöä ja tarjoavat mahdollisuuksia tuotetun koodin testaamiseen. [8.]

3 Laskentasovelluksen suunnittelu

3.1 Projektipohjaisuus

Insinöörityönä kehitettävä laskentasovellus on suunniteltu projektipohjaiseksi. Projektipohjaisuudella tarkoitetaan tässä sitä, että sovellusta käytettäessä työstettäisiin aina jotakin sovelluksen sisäistä projektia. Projekteja tulisi olla mahdollista tallentaa ja tallennettuja projekteja tulisi olla mahdollista avata sovelluksessa uudestaan. Jokainen tallennettu projekti sisältäisi omat projektikohtaiset asetuksensa ja laskentaa varten syötetyt syöttötiedot. Sovelluksen laskemaa laskentadataa ei kannata tallentaa tallennettavan projektin tietoihin, koska laskenta voidaan suorittaa yhtä hyvin aina, kun projekti avataan sovelluksessa. Laskentadata kuluttaisi turhaan tallennettavan projektin tietoihin tallennettuna tallennustavasta riippuvaa tallennuskapasiteettia.

Projektipohjainen ajattelu tuo sovelluksen kehitykseen omat haasteensa. Sovelluksessa työstettävän projektin tulisi aina nollaantua käyttäjän kirjautuessa ulos sovelluksesta. Työstettävälle projektille tulisi kehittää myös sovelluksen sisäinen tietojen tyhjennystoiminto eli mahdollisuus aloittaa uusi projekti. Käyttäjillä tulisi olla mahdollisuus käyttää sovellusta ilman, että mitään projektikohtaista dataa ikinä tallennettaisiin pysyvästi palveluntarjoajan palvelimelle. Sovelluksessa työstettävän projektin tietojen tulisi siis olla tallennettuna johonkin vain silloin, kun käyttäjä käyttää sovellusta. Sovelluksessa työstettävän projektin tiedoille tulisi kehittää väliaikainen tallennusmenetelmä, joka vastaisi edellä mainittuja vaatimuksia.

Projekteille tulisi kehittää myös pysyvä tallennusmahdollisuus. Toiveena on, että käyttäjien ei olisi välttämättä pakko tallentaa projektejaan palveluntarjoajan tietokantaan. Projektien tallennusta varten tulisi siis kehittää tietokantatallennustavan lisäksi tapa tallentaa projekteja käyttäjien omille työasemille. Tietokantatallennustavan toteuttaminen ei ole erityisen haasteellista. Projekteille tulisi toteuttaa oma tietokantarakenne, jota käyttäen projekteille luodaan oma tietokantataulu palveluntarjoajan tietokantaan. Tietokantaan tallennetut projektit kuuluisivat aina omille käyttäjilleen projektien tietoihin tallennettujen viiteavaintietojen perusteella.

Työasematallennuksen toteutus on sen sijaan haasteellisempaa. Projektin tiedot tulisi kirjoittaa tiedostoon, jonka käyttäjä voisi ladata palvelimelta omalle työasemalleen. Haasteellisen tästä tekee se, että käyttäjä voi avata tiedoston omalla työasemallaan tekstinkäsittelyohjelmalla ja tehdä siihen mielivaltaisia muokkauksia, jotka voivat toimia sovelluksen toimintaperiaatteita vastaan tai tehdä tiedoston tietorakenteesta virheellisen. Kun käyttäjä sitten lähettää virheitä sisältävän projektitiedoston omalta työasemaltaan palvelimelle sovelluksen avattavaksi, voi sovelluksessa pahimmassa tapauksessa tapahtua suoritusvirheitä. Tätä varten on siis toteutettava virheentarkistusmenetelmät, jotka projektitiedostojen tulisi palvelimelle lähetettäessä läpäistä.

Projektitiedostojen virheentarkistusominaisuutta varten tulisi tutkia menetelmiä, joissa lähetettyjä projektitiedostoja voitaisiin verrata ennalta määrättyyn malliin, jota projektitiedostojen pitäisi palvelimelle lähetettäessä vastata. Mallissa tulisi olla mahdollista määrittää projektitiedostojen tietokentille sallitut tietotyypit ja sisällöt. Projekteihin tallennettavissa syöttötiedoissa on lisäksi paljon riippuvuuksia toisiin tietoihin. Tietty tieto yhdessä tietokentässä saattaa sovelluksen toimintaperiaatteen mukaan kieltää joidenkin tietojen käyttämistä muissa tietokentissä. Tämän kaltaisia toisista tiedoista riippuvia tietojen ehdollisia tarkistuksia on tuskin mahdollista suorittaa malliin verrattaessa, vaikka projektitiedostojen vertaaminen malliin saataisiinkin toteutettua. Ehdollisia tarkistuksia varten on siis todennäköisesti kehitettävä manuaalinen virheentarkistusmenetelmä.

3.2 Laskentalogiikka

Sovellukseen ohjelmoitavan laskentalogiikan laskentakaavat ovat suhteellisen yksinkertaisia, mutta laskenta on usein taulukkomaista, joten laskennassa saatetaan käsitellä hyvin suuriakin lukumääriä kerrallaan. Laskentalogiikassa tulee myös ottaa huomi-

oon useita muuttuvia tekijöitä. Yhden arvon laskemiseen saatetaan tarvita useita viittauksia erilaisiin laskentaa ohjaaviin tietoihin ja muihin taulukkoarvoihin. Lisäksi tietyt laskentalogiikan laskentakaavat ovat käytössä useissa muissakin laskentakaavoissa. Olisikin ensisijaisen tärkeää ottaa laskentalogiikan ohjelmarakenteen suunnittelussa ja ohjelmoinnissa huomioon nämä samojen kaavojen käytöt ja muut samankaltaisuudet, jotta vältyttäisiin saman koodin esiintymiseltä useissa eri ohjelmarakenteen kohdissa. Vastaavanlaista koodin kaksoiskappaleiden esiintymistä pidetään yleisesti merkkinä huolimattomasta ohjelmointitavasta, koska se vaikeuttaa koodin ylläpidettävyyttä.

Laskentalogiikan rakentamisessa tulisi ensinnäkin käyttää abstraktioperiaatetta. Abstraktioperiaatteen mukaan jokaisen ohjelman yksittäisen toiminnallisuuden tulisi esiintyä vain yhdessä paikassa ohjelmarakennetta. Vaikka toiminnallisuuksissa olisikin pieniä eroja, olisi yleisesti ottaen hyödyllistä yhdistää kaikki samankaltaisuudet yhdeksi toteutukseksi erottaen vain eroavat toiminnallisuudet toteutuksesta. Samaa ajattelumaailmaa edustaa myös DRY-periaate (Don’t Repeat Yourself), joka menee pelkkää koodin toiston eliminointia hieman pidemmälle ottamalla huomioon myös vähemmän ilmeisempien toistomuotojen eliminoinnin. [19, s. 339; 20, s. 27.]

Olio-ohjelmoinnissa näitä periaatteita saadaan toteutettua periytymisellä. Periytymisellä tarkoitetaan käsitettä, jossa jo olemassa olevan luokan ominaisuudet voidaan antaa periytyvälle alaluokalle. Alaluokalle voidaan antaa kaikki yläluokan ominaisuudet sellaisinaan tai niistä voidaan myös syrjäyttää osia. Alaluokalle voidaan myös lisätä uusia ominaisuuksia. [21, s. 28.]

Ohjelmarakenteeseen saadaan lisättyä säännöllisyyttä myös abstrakteilla luokilla ja rajapinnoilla. Abstrakteilla luokilla tarkoitetaan luokkia, joista ei ole mahdollista luoda ilmentymää. Abstraktit luokat eivät siis vielä itsessään edusta mitään kokonaista, mutta samasta abstraktista luokasta periytyvät luokat sisältävät yhteisiä ominaisuuksia toistensa kanssa. Rajapinnat sisältävät funktioiden määrittelyjä. Rajapinnan toteuttavan luokan pitää toteuttaa rajapinnassa määritetyt funktiot, joten pelkästään rajapinnan toteuttamisen perusteella nähdään, mitä luokan on tarkoitus tehdä. [21, s. 32–34.]

Edellä mainituilla ominaisuuksilla ohjelmarakenteeseen saadaan lisättyä ohjelmointia edistävää selkeyttä ja laskentalogiikkaa voidaan pilkkoa sopivan pieniin osiin, jotta testattavuus saataisiin mahdollisimman helpoksi ja vianetsintä voitaisiin kohdistaa paremmin tiettyihin toiminnallisuuksiin. Laskentaa voitaisiin toteuttaa omiin laskentaluok-

kiinsa. Laskenta voitaisiin suorittaa niin, että ensin laskentaluokille annettaisiin kaikki laskentaan tarvittavat syöttötiedot, jonka jälkeen laskentaluokat suorittaisivat laskennan perustuen annettuihin syöttötietoihin ja palauttaisivat lasketut arvot taulukkotietotyyppien sisällä. Periytymistä hyväksikäyttämällä alemmat laskentaluokat periytyisivät ylemmistä laskentaluokista, joihin kaikkien alempien laskentaluokkien tarvitsema yhteinen laskentalogiikka olisi toteutettu.

Yhtenä ehdottomana vaatimuksena laskentalogiikan toteutustavalle oli se, että käyttäjät eivät saisi päästä lukemaan laskentalogiikan lähdekoodeja. Insinöörityön tilaaja on nähnyt suuren vaivan laskentakaavojen yhteensovittamisessa, joten ei haluta, että laskentalogiikkaan olisi vapaata pääsyä. Laskentalogiikka tulisi siis rakentaa kokonaan palvelinpuolelle, jonka lähdekoodeihin käyttäjillä ei ole pääsyä. [8.]

Varsinaista projektikohtaista laskentaa on tarkoitus suorittaa useiden sovelluksen syöttömoduulien kautta. Kaikki saman syöttömoduulin kautta suoritettavat laskut eivät kuitenkaan noudata samaa laskentalogiikkaa, koska syöttömoduuleissa pystytään tekemään laskennan kulkuun vaikuttavia valintoja. Osa näistä valinnoista on myös riippuvaisia toisista valinnoista. Nämä riippuvuudet voivat olla esimerkiksi sellaisia, että tietyn arvon valitseminen yhdestä valintakentästä saattaa antaa vain tietyt valintamahdollisuudet joihinkin muihin valintakenttiin tai valinta saattaa asettaa muita valintakenttiä kokonaan epäaktiivisiksi, jolloin ne eivät ota osaa suoritettavaan laskentaan millään tavalla. Osa syöttömoduuleissa olevista valintamahdollisuuksista on myös määritetty jo ennen syöttömoduulisivun avaamista.

Monimutkaisuutta sovelluksen kehittämiseen tuo useiden aukinaisten välilehtien välinen käyttö. Oletetaan tilanne, jossa käyttäjällä on auki kaksi sovelluksen eri sivua selaimen eri välilehdillä. Ensimmäisen välilehden sivulla on tietoja, jotka määrittävät toisen välilehden sivun valintamahdollisuuksia. Toisen välilehden sivu on tässä tapauksessa yksi sovelluksen monista syöttömoduuleista. Jos käyttäjä nyt molempien välilehtien ollessa auki vaihtaa ensimmäisen välilehden sivulla projektin asetuksia, jotka määrittävät toisen välilehden sivun valintamahdollisuuksia, kohtaamme ongelman. Toisella välilehdellä eli syöttömoduulisivulla on nyt päivittämättömiä valintamahdollisuuksia, joille pitäisi tehdä jonkinlainen tarkistus ennen laskennan suorittamista. Jos käyttäjä yrittää suorittaa laskentaa päivittämättömillä valinnoilla, tulisi käyttäjälle ilmoittaa valintojen olevan päivittämättömät ja tuoda uudet päivitetyt valintamahdollisuudet sivulle.

3.3 Raportointi

Sovelluksen varsinainen käyttötarkoitus on koostaa raportteja syöttömoduulien kautta lisätyistä syöttötiedoista. Raporttien koostamista varten luodaan eräänlainen tulostaulukko kaikista lasketuista arvoista. Tulostaulukko tulee olemaan kokonaan piilotettu käyttäjiltä. Tulostaulukkoa tarvitaan vasta, kun käyttäjä haluaa koostaa itselleen raportin, joten tulostaulukko on parempi luoda aina erikseen raportteja tarvittaessa. Tulostaulukkoa ei siis tarvitse tallentaa mihinkään tietovarastoon edes hetkellisesti. Raporttimalleja on useita, mutta niistäkin osa sisältää samaa logiikkaa toisten raporttimallien kanssa. Ohjelmarakennetta suunnitellessa tulisi taas ottaa nämä samankaltaisuudet huomioon.

Raportteja tulisi pystyä ensisijaisesti lukemaan suoraan selaimella. Tällöin raporttien tiedot tuotaisiin esille HTML-merkkauskieltä käyttäen. Raportteja tulisi myös pystyä tallentamaan omalle työasemalle kahdessa eri tiedostoformaatissa. Ensimmäisen tiedostoformaatin tulisi olla sellainen, että raporttien tiedot pystyttäisiin tuomaan eri taulukkolaskentaohjelmiin taulukkolaskentaohjelmien omilla tuontitoiminnoilla. Näin ollen käyttäjät pystyisivät analysoimaan raporttien tuloksia taulukkolaskentaohjelmien sisällä. Tähän tarkoitukseen oli jo määritetty CSV-tiedostoformaatin (Comma-Separated Values) käyttö. Comma-Separated Values tarkoittaa suomennettuna pilkulla erotettuja arvoja, mutta erottimena voidaan käyttää myös esimerkiksi puolipistettä (esimerkkikoodi 1). [22.] Pilkkua tarvitaan taulukkolaskentaohjelmissa lukujen desimaalien erottamiseen, joten puolipisteen käyttö tietojen erottimena on tässä tapauksessa suotavaa.

```
;Yläotsikko 1;Yläotsikko 2;Yläotsikko 3
Sivuotsikko 1;1,1;2,2;3,3
Sivuotsikko 2;4,4;5,5;6,6
Sivuotsikko 3;7,7;8,8;9,9
```

Esimerkkikoodi 1. Puolipiste CSV-tiedoston sisältämien tietojen erottimena.

Puolipisteellä erotetut tiedot sijoittuvat taulukkolaskentaohjelmiin tuotaessa omille soluilleen (kuva 2), jos tuontitoiminnon yhteydessä määritetään, että tietojen erottimena on käytetty puolipistettä. Mahdollisissa otsikkotiedoissa on otettava huomioon, ettei erikoismerkkien käyttö aiheuta tietojen virheellistä asettumista taulukkolaskentaohjelman soluihin tuontitoimintoa käyttäessä.

	A	B	C	D
1		Yläotsikko 1	Yläotsikko 2	Yläotsikko 3
2	Sivuotsikko 1	1,1	2,2	3,3
3	Sivuotsikko 2	4,4	5,5	6,6
4	Sivuotsikko 3	7,7	8,8	9,9

Kuva 2. Taulukkolaskentaohjelmaan tuotu esimerkkikoodin 1 mukainen CSV-tiedosto.

Toisena tiedostoformaattina raporttien koostamisessa haluttiin käyttää PDF-tiedostoformaattia (Portable Document Format). Tämä mahdollistaisi sen, että koostettuja raportteja voitaisiin tallentaa omalle työasemalle siistimmän näköisinä ja avata PDF-tiedostojen lukemiseen soveltuvilla ohjelmistoilla. Lisäksi raportteja olisi mahdollista tulostaa paperille pääosin samannäköisinä. [23.] Toteutustavassa tulisi miettiä, miten tarvittava data saataisiin parhaiten siirrettyä PDF-tiedostoformaattiin. PDF-dokumenttien ulkoasuun ei vielä toistaiseksi kiinnitetä erityistä huomiota. Pääasia, että PDF-dokumenttien luominen saadaan toimimaan ja kaikki vaadittavat tiedot saadaan näkymään luotuihin dokumentteihin.

Raportteja varten suoritetaan sama laskenta riippumatta luotavan raportin tyypistä. HTML-raporteissa tiedot yksinkertaisesti näytetään sivulla näkyvässä taulukossa, joten vastuu tietojen esittämisestä siirretään näkymän luomisen yhteyteen. CSV-raporteissa tiedoista luodaan tietynlainen merkkijono noudattaen CSV-tiedostoformaatin sääntöjä. Luotu merkkijono kirjoitetaan tiedostoon, jonka käyttäjä lataa työasemalleen. HTML- ja CSV-raporttien toteutustavat ovat hyvin selkeitä jo suunnitteluvaiheessa. PDF-raporttien luominen vaatii sen sijaan tarkempaa tarkastelua, koska PDF-raporttien luomiseen tarvitaan erillistä työkalua.

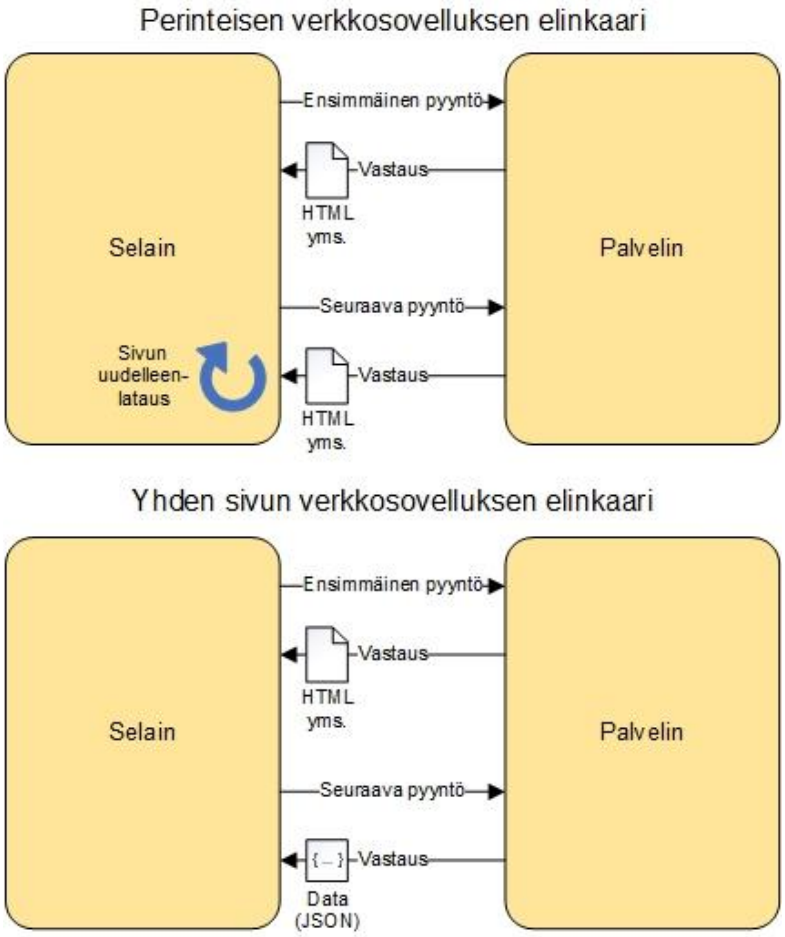
3.4 Sujuvan käyttökokemuksen takaaminen

Sovelluksen käyttö perustuu osittain jatkuvaan iterointiin halutun lopputuloksen saavuttamiseksi. Käytännössä tämä tarkoittaa sitä, että käyttäjät saattavat antaa sovellukselle hyvinkin pienellä aikavälillä useita eri syöttötietoja laskentaa varten. Syöttötietoja saatetaan myös poistaa heti, jos niiden tuottamat lopputulokset eivät ole miellyttäviä. Sovellus tulee siis olemaan hyvin käyttäjäläheinen, jonka vuoksi sovelluksen käyttökokemuksesta on saatava yhtenäinen ja sujuva.

Verkkosivut toimivat tavallisesti niin, että koko sivu uudelleenladataan aina pyyntöjen yhteydessä. Selain saattaa näyttää sivunlatauksen yhteydessä hetkellisesti tyhjää sivua ja käyttäjät saattavat kokea lyhyetkin odotusajat epämiellyttävänä. Jos sivun käyttö ei sisällä jatkuvaa vuorovaikutusta sen käyttäjien kanssa, ei sivunlatausten odottaminen todennäköisesti muodostu sen suuremmaksi ongelmaksi. Enemmän vuorovaikutusta sisältävien sovellusten tapauksessa ylimääräiset odotusajat tulisi kuitenkin saada mahdollisimman lyhyiksi. [24; 25.]

Ajax-pyynnöt mahdollistavat selaimen ja palvelimen välisen vuorovaikutuksen ilman kokonaisia sivunlatauksia. Ajax-pyynnöillä palvelimelta haetaan yleensä pelkkää dataa, joka ei sisällä ylimääräisiä resursseja, kuten HTML-merkkausta. Sivu päivitetään Ajax-pyynnön jälkeen dynaamisesti haetun datan avulla. Data palautetaan palvelimelta tavanomaisesti JSON-formaatissa (JavaScript Object Notation). Datan haku Ajax-pyynnöllä tapahtuu nopeasti ja on käytännössä huomaamaton toimenpide käyttäjälle. [24; 25; 26.]

Ajax-teknologiat ja parantunut työkalutuki ovat edesauttaneet SPA-arkkitehtuuria (Single-Page Application) noudattavien yhden HTML-sivun varaan rakennettujen verkkosovellusten yleistymistä ja suosion kasvua. SPAt ovat verkkosovelluksia, joissa kaikki tarvittava asiakaspuolen koodi (HTML, CSS ja JavaScript) ja muut resurssit haetaan yhdellä sivunlatauksella ja kaikki sivun päivitysoperaatiot suoritetaan Ajax-pyynnöillä. Näin käyttökokemuksesta saadaan sujuvampi, koska kokonaiset sivunlataukset saadaan kokonaan pois sovelluksen elinkaaresta (kuva 3). Yhden sivun sovellukset siirtävät Ajax-pyyntöjen lisäksi sovelluksen logiikkaa yhä enemmän käyttäjien selainten suoritettaviksi. Myös siirtymät eri sivujen välillä voidaan toteuttaa niin, että palvelimelta haetaan pelkästään sivulla näytettävää dataa, jolloin käyttäjien selaimet huolehtivat kokonaan uuden sivun näkymän päivittämisestä. Tarve ladata pelkästään pieniä datamääriä kerrallaan tekee siirtymistä näkymien välillä kevyitä, koska ainoastaan uudelle näkymälle välttämätön tieto tarvitsee siirtää verkossa. Näin käyttökokemuksesta saadaan vuorovaikutteisempi ja sovelluksesta kokonaisuudessaan käytettävämpi. [24; 25.]



Kuva 3. Perinteisen verkkosovelluksen elinkaari verrattuna yhden sivun verkkosovelluksen elinkaareen [24].

Laskentasovellus tullaan toteuttamaan sujuvan käyttökokemuksen takaamiseksi yhden sivun verkkosovelluksena. Sovelluksen syöttömoduuleissa syötetyt syöttötiedot lähetetään aina palvelimelle toteutetun laskentalogiikan laskettaviksi ja syötetyt tiedot tallennetaan niille tarkoitettuihin tietovarastoihin. Tämän jälkeen syöttötietojen avulla lasketut tulokset palautetaan selaimelle ja näytetään sivulla. Vastaavasti annettuja syöttötietoja voidaan myös poistaa tietovarastosta Ajax-pyyntöjen avulla. Toteutustapa on siis käytännössä hyvin suoraviivainen ja käyttäjät pystyvät olemaan hyvinkin aktiivisesti vuorovaikutuksessa sovelluksen kanssa sovelluksen toiminnan ja käyttökokemuksen pysyessä silti sujuvana.

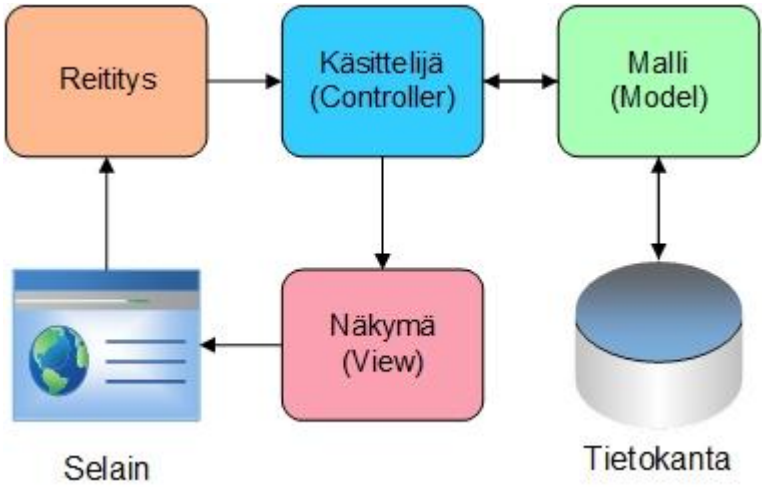
4 Laskentasovelluksen toteutus

4.1 Sovelluksen runko

Laskentasovelluksen palvelinpuolen ohjelmointi tullaan toteuttamaan PHP-ohjelmointikielellä aiemman kokemukseni ja PHP:n suosioon perustuvan valtavan lähdeaineiston ja tuen vuoksi. Sovelluksen runkona tullaan käyttämään avoimeen lähdekoodiin perustuvaa PHP-ohjelmointikielellä toteutettua Laravel-verkkosovelluskehystä, jota tukee aktiivinen ja kasvava kehittäjäyhteisö. Sen avulla tavanomaisimmat verkkosovellusten vaatimat toiminnallisuudet saadaan toteutettua valmiita ratkaisuja käyttäen helposti ja nopeasti. [27.]

Perusteita Laravelin valintaan olivat mm. kattava dokumentaatio, opetusmateriaalin paljous ja selkeä rakenne. Laravel tähtää siihen, että sovelluskehitysprosessista saataisiin mahdollisimman miellyttävä, mutta kuitenkin niin, ettei sovelluksen toiminnallisuus kärsisi. Laravel käyttää muutamia muiden verkkosovelluskehysten valmiita toiminnallisuuksia, joten täysin omaan varaansa sitä ei ole rakennettu. Laravelin matalaa oppimiskynnystä pidetään yhtenä sen suurena vahvuutena. [27.]

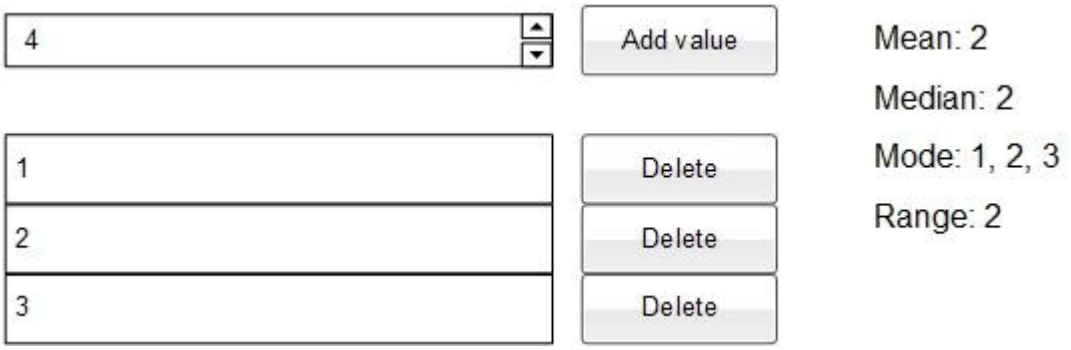
Laravel perustuu MVC-ohjelmistoarkkitehtuuriin (Model-View-Controller), jonka tarkoituksena on sovelluksen käyttöliittymän erottaminen sovelluslogiikasta (kuva 4). MVC-arkkitehtuuri on suosittu suunnittelumalli verkkosovelluskehityksessä. MVC-arkkitehtuuri perustuu nimensä mukaisesti kolmeen komponenttiin, jotka ovat model eli malli, view eli näkymä ja controller eli käsittelijä. Mallit kuvaavat sovellukseen liittyviä loogisia toimintoja ja tietoja. Malleihin liittyvät tiedot ovat usein tallennettuna sovelluksen käyttämään tietokantaan. Näkymät ovat sovelluksen visuaalisia esitysmuotoja, jotka esittävät usein malleihin liittyviä tietoja käyttäjälle HTML-merkkauskielen avulla. Käsittelijät ohjaavat ohjelman suoritusta mallien ja näkymien välillä vastaanottamalla käyttäjiltä tulevia käskyjä ja määrittämällä niiden perusteella tehtäviä toimenpiteitä. [28.]



Kuva 4. MVC-arkkitehtuurin toimintaperiaate Laravelissa [28].

Normaalisti Laravel toteuttaa MVC-arkkitehtuuria niin, että sovelluksen käyttäjä lähettää selaimella pyynnön palvelimelle, jolloin pyyntö ohjataan Laravelin reittienmääritystiedostossa määritetyn reitin kautta useimmiten reittiä varten toteutetulle käsittelijälle. Tavallisesti käsittelijä käsittelee jotakin sovellukseen määritettyä mallia, jonka tiedot haetaan yleensä tietokannasta. Mallin käsittelyn jälkeen käsittelijä palauttaa käyttäjän selaimeen näkymän, joka sisältää näkymästä riippuen resursseja, kuten HTML-merkkauskieltä, CSS-tyyliohjeita ja kuvia. Yhden sivun verkkosovellusten tapauksessa käsittelijästä kuitenkin useimmiten palautetaan asiakaspuolelle pelkkää dataa kokonaisen näkymän sijaan ja näkymän esittäminen siirretään asiakaspuolen hoidettavaksi. [24; 28.]

Oletetaan, että ollaan kehittämässä yksinkertaista tilastollisia lukuja laskevaa laskentasovellusta, joka laskisi käyttäjän syöttämien lukujen keskiarvon, mediaanin, moodin ja vaihteluvälin pituuden. Lukuja voitaisiin syöttää sivulla näkyvään taulukkoon yksi kerrallaan ja syötettyjä lukuja voitaisiin myös poistaa yksi kerrallaan. Kuvassa 5 on esitetty tämän esimerkkinä toteutettavan laskentasovelluksen toimintaperiaatteen havainnollistamiseksi rautalankamalli siitä, miltä sovelluksen käyttöliittymä voisi suurin piirtein näyttää.



Kuva 5. Rautalankamalli esimerkkinä toteutettavan laskentasovelluksen käyttöliittymästä.

Luvun syötön ja poistamisen yhteydessä sivulle päivitettäisiin suoraan uudet tilastolukuja varten lasketut arvot. Syötetyt luvut myös tallennettaisiin johonkin, jolloin tallennetut luvut ja lasketut arvot näytettäisiin sivulla vielä sivun sulkemisen jälkeenkin. Esimerkkikoodissa 2 on esitetty, miltä Laravelin reittienmääritystiedoston sisältö voisi tällaisen laskentasovelluksen tapauksessa näyttää.

```
<?php

Route::group(array('prefix' => 'api'), function()
{
  Route::resource('statistics', 'StatisticsController',
             array('only' => array('index', 'store', 'destroy')));
});

App::missing(function()
{
  return View::make('index');
});
```

Esimerkkikoodi 2. Toteutettavaa esimerkkisovellusta varten määritetyt reitityssäännöt.

Esimerkkikoodin 2 alussa on käytetty sanaa Route, joka on Laravelin facade-luokka eli eräänlainen julkisivuluokka. Laravelin julkisivuluokat mahdollistavat luokkien funktioiden käyttämisen staattisella syntaksilla eli kahta peräkkäistä kaksoispistettä käyttäen ilman uuden luokan alustamista new-sanalla. Kyseessä ei siis kuitenkaan ole luokka, joka sisältäisi staattisia funktioita. Oikeasti luokat haetaan julkisivuluokkia käytettäessä Laravelin IoC (Inversion of Control) containerista eli käänteisen hallinnan säiliöstä, joka on tarkoitettu riippuvuusluokkien injektointiin (Dependency Injection). Julkisivuluokkien tarkoituksena on tarjota sovelluskehittäjälle ytimekäs ja ilmaiseva syntaksi luokkien käyttöön, mutta kuitenkin niin, että koodi pysyy testattavana. [29; 30.]

Route-luokan ensimmäinen käytettävä funktio esimerkkikoodin 2 tapauksessa on group, jonka avulla reitityssääntöjä saadaan ryhmiteltyä yhteen sivun osoitepoluissa olevien etuliitteiden perusteella käyttämällä prefix-sanaa ja sille annettavaa etuliitettä. Esimerkin reittipoluissa on käytetty api-etuliitettä (Application Programming Interface), joka kuvastaa reitin kuuluvuutta Ajax-pyyntöjä varten toteutettavaan ohjelmointirajapintaan. Vastaavasti reiteille voidaan lisätä myös suodatusta. Jos esimerkiksi halutaan, että vain kirjautuneella käyttäjällä on pääsy tiettyihin resursseihin, voidaan käyttää prefix-sanan kaltaisesti before-sanaa ja antaa sille käytettävän suodattimen nimi, joka voisi olla käyttäjän todentamisessa esimerkiksi auth. Itse suodattimelle kirjoitetun funktion sisällä voidaan määrittää jatkotoimenpiteet sen varalle, ettei käyttäjä olekaan kirjautunut sovellukseen. [31; 32.]

Esimerkkikoodissa 2 on määritetty kolme reittiä Route-luokan resource-funktiolla, jonka avulla sovellukseen saadaan luotua REST-arkkitehtuuria (Representational State Transfer) noudattavia reitityssääntöjä. REST-arkkitehtuuri tähtää mahdollisimman yksiselitteisten, ylläpidettävien ja skaalautuvien sovelluspalveluiden kehitykseen, johon HTTP-protokolla tarjoaa avuksi rajapinnan. REST-arkkitehtuuria noudattavaa verkkopalvelua kuvataan termillä RESTful, joka toimii kuvauksessa adjektiivina. Taulukossa 1 on kuvattu, miten Laravel luo Route-luokan resource-funktiolla REST-arkkitehtuuria noudattavia reitityssääntöjä. Esimerkkikoodin 2 tapauksessa resurssin nimeksi on määritetty statistics ja resurssien käsittelijäksi on määritetty StatisticsController. Funktiolle annettavan array-tietotyypin sisällä määritetään only-sanaa käyttäen, että haluamme käsittelijän vastaavan vain taulukossa 1 esiintyviin index-, store- ja destroy-toimintoihin. Laravelissa on myös mahdollista määrittää reitityssääntöjä yksittäisille osoitteille resource-funktion käytön sijaan. [33; 34.]

Taulukko 1. Laravelin resurssikäsittelijöiden noudattamat toiminnot reittejä kohden [33].

HTTP-pyyntö	**Reitin polku**	**Käsittelijän funktion nimi**
GET	/resource	index
GET	/resource/create	create
POST	/resource	store
GET	/resource/{id}	show
GET	/resource/{id}/edit	edit
PUT/PATCH	/resource/{id}	update
DELETE	/resource/{id}	destroy

Sivulle tultaessa tallennetut luvut ja lasketut arvot on tarkoitus hakea Ajax-pyynnöllä käyttäen reittiä /api/statistics. Pyyntöön käytetään datan pyytämiseen tarkoitettua

HTTP-protokollan GET-pyyntöä, joka viedään StatisticsController-käsittelijän index-funktiolle. Luvun lisäämisessä taulukkoon käytetään samaa reittipolkua, mutta pyyntönä käytetään HTTP-protokollan POST-pyyntöä, jonka avulla pyynnön yhteyteen voidaan lisätä sisältöä. Luvun lisäämiseen käytetään käsittelijän store-funktiota. Kun yksittäinen luku halutaan poistaa lisättyjen lukujen taulukosta, voidaan käyttää esimerkiksi polkua /api/statistics/5, jossa luku 5 kuvastaisi yksittäistä lukua yksilöivää tunnistetta. Luvun poistamiseen käytetään HTTP-protokollan DELETE-pyyntöä, joka on tarkoitettu nimensä mukaisesti palvelimella sijaitsevien resurssien poistamiseen. Yksinkertaisuuden vuoksi esimerkkiin on määritetty vain nämä kolme vaadittua reittiä. Jos resurssit olisivat yksittäisten lukujen sijaan jotakin suurempaa tietomäärää kuvaavia malleja, kuten esimerkiksi blogijulkaisuja, saattaisi olla viisasta toteuttaa kaikki taulukkoa 1 vastaavat reititystoiminnot, jolloin yksittäisiä blogijulkaisuja voitaisiin myös tarkastella ja muokata omilla sivuillaan. [33.]

Esimerkkikoodin 2 lopussa käytetään App-luokan missing-funktiota. Se on tarkoitettu virheenkäsittelyyn, joka suoritetaan määrittämättömien reittien käytön yhteydessä. Edes sovelluksen etusivulle ei ole esimerkin tapauksessa määritetty reittiä, joten yrittäessä avata etusivua, tapahtuisi reititys missing-funktiolle määritettyjen sääntöjen mukaan. Esimerkkikoodissa 2 missing-funktion sisällä on käytetty View-luokan make-funktiota, jonka avulla selaimelle saadaan palautettua jokin näkymä. Esimerkkikoodin 2 tapauksessa selaimelle palautetaan index-niminen näkymä, joka sijaitsisi oletusarvoisesti Laravelin näkymäkansion sisällä olevassa tiedostossa index.php. Näkymän palauttamisen jälkeen kaikki sivulla tapahtuva reititys osoitteesta toiseen tapahtuisi yhden sivun sovelluksen tapauksessa asiakaspuolella ja palvelimeen oltaisiin yhteydessä vain Ajax-pyynnöillä dataa hakiessa. [35; 36.]

Pyynnöt esimerkkikoodin 2 resource-funktiolla määritettyihin reitteihin ohjattaisiin StatisticsController-nimiselle käsittelijäluokalle, joka on esitetty kokonaisuudessaan esimerkkikoodissa 3. Käsittelijäluokka StatisticsController periytyy Laravelin omasta BaseController-luokasta. BaseController-luokka sisältää toteutuksia, joita tarvitaan kaikissa käsittelijäluokissa, joten periytyminen BaseController-luokasta on välttämätöntä. Luokan konstruktori eli luokan alustuksen yhteydessä suoritettava funktio saa parametreina luokan, joka vastaa syötettyjen lukujen tallennuksesta ja luokan, joka suorittaa tilastolukujen laskemisen. Kyseiset luokat otetaan käyttöön use-sanalla luokkien nimiavaruuksien perusteella, jotka on määritetty vastaamaan esimerkkikoodin 3 tapauksessa luokkien kansiopolkua sovelluksessa. [33.]

```
<?php

use StatisticsCalculator\ValueRepositoryInterface;
use StatisticsCalculator\Calculators\StatisticsCalculator;

class StatisticsController extends BaseController {

  public function   construct(ValueRepositoryInterface $repository,
                              StatisticsCalculator $calculator)
  {
    $this->repository = $repository;
    $this->calculator = $calculator;
  }

  public function index()
  {
    $data = array();
    $data['values'] = $this->repository->all();
    $data['calculated'] = $this->calculator
                              ->setValues($data['values'])
                              ->calculate();

    return Response::json($data);
  }

  public function store()
  {
    $this->repository->create(Input::get('value'));

    return $this->index();
  }

  public function destroy($id)
  {
    $this->repository->destroy($id);

    return $this->index();
  }

}
```

Esimerkkikoodi 3. Esimerkkisovellusta varten toteutettu käsittelijäluokka kokonaisuudessaan.

Käsittelijäluokan konstruktorin parametreissa on nähtävissä ValueRepositoryInterface-niminen rajapinta. Tässä käytetään tietojen tallennukseen kehitettyä suunnittelumallia, jota kutsutaan säilömalliksi (Repository Pattern). Todellisuudessa käsittelijäluokan konstruktorille annetaan Laravelin käänteisen hallinnan säiliötä käyttäen riippuvuusinjektiona säilöluokka, joka toteuttaa ValueRepositoryInterface-nimisen rajapinnan. Kyseinen rajapinta on sidottu käytettävään säilöluokkaan sovelluskohtaisissa asetuksissa, jotka on määritetty käsittelijäluokan ulkopuolella. Säilömallin ajatuksena on, ettei käsittelijäluokan tarvitse tietää varsinaisen tallennusmenetelmän toteutustavasta mitään. Jos tallennusmenetelmää halutaan myöhemmin vaihtaa, voidaan kehittää uusi säilöluokka ja sitoa käytössä oleva rajapinta uuteen kehitettyyn luokkaan. Käsittelijäluok-

kaan ei siis tarvitse tehdä minkäänlaisia muutoksia, vaikka tallennusmenetelmä muuttuisikin. [37.]

Käsittelijäluokka sisältää kolme reiteille määritettyä funktiota, jotka ovat index, store ja destroy. Funktio index on tarkoitettu kutsuttavaksi aina, kun asiakaspuolelle tarvitsee päivittää uusimmat tiedot. Sitä kutsutaan sivun avaamisen yhteydessä, luvun lisäämisen yhteydessä ja luvun poistamisen yhteydessä. Näin ollen sitä kutsutaan myös storeja destroy-funktioiden suorittamisen päätteeksi. Funktio index palauttaa asiakaspuolelle kaikki säilössä olevat käyttäjän syöttämät luvut ja laskentaluokan laskemat arvot tilastoluvuille. Täten asiakaspuoli saadaan pidettyä ajan tasalla jokaisen käyttäjän suorittaman tapahtuman jälkeen.

Tarve laskea lukuja erikseen jokaisen toiminnon yhteydessä on eräänlainen erikoistapaus. Tavallisesti vain käsiteltäisiin joitakin olemassa olevia resursseja, kuten blogijulkaisuja, jolloin ne palautettaisiin asiakaspuolelle sellaisinaan. Laskettuja lukuja ei kuitenkaan kannata tallentaa mihinkään pysyvästi, koska tallennettuina ne veisivät turhaan tallennuskapasiteettia ja luvut voidaan kuitenkin laskea aina erikseen sivun alustamisen ja päivittämisen yhteydessä.

Tiedot palautetaan asiakaspuolelle JSON-formaatissa käyttäen Laravelin Responseluokan json-funktiota, kuten esimerkkikoodista 3 nähdään. Käsittelijäluokan storefunktion yhteydessä käyttäjän syöttämä luku saadaan käyttöön Laravelin Input-luokan get-funktiolla, joka hakee käyttäjän syöttämän luvun HTTP-protokollan POST-pyynnölle annetuista tiedoista. Käsittelijäluokan destroy-funktio saa parametrina poistettavan luvun tunnisteen, joka annetaan käytettävän HTTP-protokollan DELETE-pyynnön polussa. Säilöluokan destroy-funktio tulisi toteuttaa niin, että tunnistetta vastaava luku poistetaan, jos sellainen säilöstä löytyy. [36; 38.]

Esimerkkinä toteutettavaan laskentasovellukseen saatiin edellisten esimerkkien mukaisesti toteutettua ohjelmointirajapinta asiakaspuolelta suoritettavia Ajax-pyyntöjä varten hyödyntäen Laravelin toteutuksia. Samoja toimintaperiaatteita voidaan käyttää myös insinöörityönä kehitettävän laskentasovelluksen toteuttamiseen. Pyynnöt ohjataan reitityssääntöjen mukaisesti niille tarkoitetuille käsittelijöille ja käsittelijät käyttävät säilöluokkia mallien käsittelyyn. Säilömallia käyttäessä käsittelijä ei itse tiedä mallien tallennusmenetelmien toteutustavoista, koska tallennusmenetelmät on toteutettu säilöluokan sisällä. Lisäksi esimerkin tapauksessa käsittelijässä käytetään laskentaluokkaa tilasto-

lukujen laskemiseen, jolloin sovelluksen logiikkaa saadaan pilkottua yhä pienempiin osiin, joka taas edesauttaa koodin ylläpidettävyyttä ja testattavuutta. Laravelin noudattama MVC-arkkitehtuuri ei toteudu yhden sivun sovellusten tapauksessa aivan sellaisenaan, koska Laravelin käsittelijöistä palautetaan asiakaspuolelle tekstimuotoista dataa JSON-formaatissa kokonaisten näkymien sijaan.

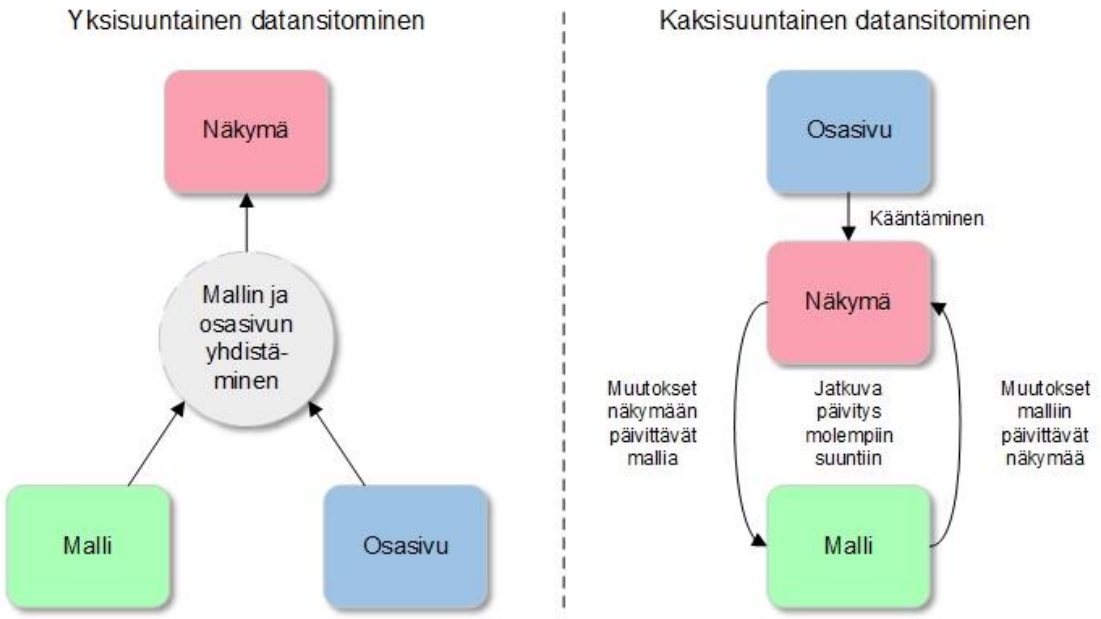
Insinöörityönä toteutettavan laskentasovelluksen tapauksessa tuli vielä tarkistaa, että sivulla näkyvät valintavaihtoehdot ovat päivitetyt ennen laskennan suorittamista. Tilastolukuja laskevan esimerkin tapauksessa sivulla ei ole laskentaa ohjaavia valintavaihtoehtoja, jotka olisi mahdollista määrittää sovelluksen toisella sivulla sijaitsevissa asetuksissa. Tätä varten joudutaan kuitenkin insinöörityönä kehitettävän laskentasovelluksen tapauksessa toteuttamaan Ajax-pyynnön yhteydessä tapahtuva virheentarkistus, jossa aukinaisen sivun tietoja verrataan palvelimelle tallennettuihin päivitettyihin tietoihin ennen laskennan suorittamista.

4.2 Asiakaspuolen ohjelmointi

Insinöörityönä kehitettävän laskentasovelluksen asiakaspuolen ohjelmointi tullaan toteuttamaan avoimeen lähdekoodiin perustuvan JavaScript-ohjelmointikielellä toteutetun AngularJS-sovelluskehyksen avulla, jota käyttäen sovelluksesta saadaan tehtyä SPA-arkkitehtuuria noudattava yhden sivun verkkosovellus. AngularJS valittiin laskentasovelluksen kehitykseen sen ylivoimaisen yksinkertaisuuden ja helppouden vuoksi, mutta se kykenee ehdottomasti myös vastaamaan kaikkiin laskentasovelluksen vaatimuksiin ongelmitta. [39.]

AngularJS mahdollistaa jokseenkin MVC-arkkitehtuurin kaltaisen arkkitehtuurimallin käytön myös asiakaspuolella, joten näin ollen asiakaspuolella käytettävien mallien käsittelylogiikka ja muu yrityslogiikka saadaan erotettua dokumentin esityslogiikasta. AngularJS:n avulla asiakaspuolella voidaan käyttää omaa osoitepolkuja noudattavaa reititintä, jonka avulla koodien suoritusta voidaan ohjata omille asiakaspuolen käsittelijöilleen ja näyttää reitille määritetyt HTML-osasivut. AngularJS vaihtaa reititintä käyttäessään yhden ja saman auki olevan dokumentin osasivuja reittien perusteella, jolloin koko dokumenttia ei tarvitse koskaan uudelleenladata. [39.]

Yksi AngularJS:n hienoista ominaisuuksista on kaksisuuntainen datansitominen, jonka avulla malleihin tehdyt muutokset saadaan näkyviin esillä olevan dokumentin näkymään automaattisesti (kuva 6). Sama toimii myös toiseen suuntaan eli käyttäjän tehdessä muutoksia dokumentin näkymään, päivittyvät muutokset suoraan myös malleihin. Näin ollen sovelluskehittäjä pääsee paljon helpommalla, koska kehittäjän ei tarvitse itse huolehtia HTML-dokumentin käsittelystä DOM-ohjelmointirajapintaa käyttäen, koska AngularJS tekee sen kehittäjän puolesta. [40.]



Kuva 6. Yleisemmin käytetty yksisuuntainen datansitominen verrattuna AngularJS:n kaksisuuntaiseen datansitomiseen [40].

Seuraavaksi jatketaan aiemmin aloitetun tilastolukuja laskevan laskentasovellusesimerkin kehittämistä. AngularJS-sovelluksen kehitys aloitetaan AngularJS-moduulin luomisella ja sen konfiguroimisella (esimerkkikoodi 4). AngularJS:n moduulit ovat eräänlaisia säiliöitä AngularJS-sovelluksen eri osille, joita ovat mm. controllerit eli käsittelijät, servicet eli palvelut, filterit eli suodattimet ja directivet eli direktiivit. [41.]

```
var app = angular.module('statisticsCalculator', ['ngRoute']);

app.config(function($routeProvider) {
  $routeProvider.
    when('/', {
      template: 'Hello!'
    }).
    when('/statistics', {
      templateUrl: 'templates/statistics.html',
      controller: 'StatisticsCtrl',
```

```
      resolve: {
        statisticsData: function(statisticsService) {
          return statisticsService.getData().then(function(response) {
            return response.data;
          }, function() {
            // handle error
          });
        }
      }
    }).
    otherwise({
      redirectTo: '/'
    });
});
```

Esimerkkikoodi 4. AngularJS-moduulin luominen ja sen konfigurointi.

Esimerkkikoodin 4 alussa alustettavaan app-muuttujaan asetetaan AngularJS-moduuli, jolle annetaan nimeksi statisticsCalculator. Moduuleja voidaan luoda useampia, joten sen vuoksi moduulit on aina nimettävä. Moduulille annetaan myös argumenttina taulukko, johon on mahdollista määrittää moduulissa käytettäviä riippuvuuksia. Esimerkkikoodissa 4 on määritetty, että sovelluksessa halutaan käyttää AngularJS:n ngRoute-moduulia, joka mahdollistaa yksinkertaisen asiakaspuolen reitityksen toteuttamisen AngularJS-sovellusta varten. [41; 42.]

Esimerkkikoodissa 4 kutsutaan luodun app-moduulimuuttujan config-funktiota, jolle annetaan argumenttina sovelluskohtaiset konfiguraatiot suorittava funktio. Funktio saa parametrina AngularJS:n riippuvuusinjektio-ominaisuutta käyttäen ngRoute-moduulin osaksi kuuluvan $routeProvider-palvelukomponentin. Sen sisältämää when-funktiota käyttäen saadaan määritettyä reititysääntöjä AngularJS-sovellukseen. [41; 43.]

Ensimmäinen esimerkkikoodissa 4 määritetty reitti on sovelluksen etusivu, jossa näytetään pelkästään teksti ”Hello!”. Etusivulle ei ole määritetty käsittelijää, koska etusivu ei sisällä mitään erityistä logiikkaa. Toinen määritetty reitti on tarkoitettu tilastolukuja laskevalle sivulle, jota varten aiemmin toteutettiin palvelinpuolen logiikkaa. Sivua varten on tehty HTML-osasivu, jonka sijainti kerrotaan templateUrl-sanaa käyttäen. Sivua varten on myös määritetty erillinen käsittelijä, jolle on annettu nimeksi StatisticsCtrl. Käsittelijä voitaisiin välittää reitille myös funktiona, mutta esimerkin tapauksessa sille on annettu nimi, joka kertoo käsittelijän logiikan sijaitsevan jossakin muualla. Lisäksi resolve-sanalla saadaan määrättyä, että sivua ei näytetä ennen palvelinpuolelta tarvittavan datan hakemista. Lopuksi $routeProvider-komponentin otherwise-funktiota käyttäen luodaan sääntö, jonka mukaan käyttäjä ohjataan etusivulle jos hän yrittää navigoida

sivulle, jolle ei ole luotu reitityssääntöä. [43.] Laadukkaan verkkosovelluksen tapauksessa tässäkin tapauksessa tulisi mielellään jollakin tavalla ilmoittaa käyttäjälle, että hän yritti navigoida määrittämättömälle sivulle.

Tilastolukuja laskevaa sivua varten tarvittava data haetaan palvelimelta asynkronisesti statisticsService-palvelun sisällä käytettävän AngularJS:n oman $http-palvelun avulla (esimerkkikoodi 5). AngularJS:n $http-palvelun avulla tehtävä Ajax-pyyntö palauttaa HttpPromise-objektin, jonka sisältämää then-funktiota kutsutaan, kun vastaus saadaan. Jos Ajax-pyyntö onnistui, palautetaan saatu data käsittelijän käyttöön then-funktiolle ensimmäisenä argumenttina annettavan funktion kautta. Ajax-pyynnön epäonnistumista varten voidaan määrittää virheenkäsittelyä then-funktion toiseksi argumentiksi annettavassa funktiossa. [44.] Virheenkäsittelyä voisi olla esimerkiksi siirtyminen omalle virheilmoituksen näyttävälle sivulle.

```
app.factory('statisticsService', function($http) {
  return {
    getData: function() {
      return $http.get('/api/statistics');
    },
    addValue: function(value) {
      return $http.post('/api/statistics', { value: value });
    },
    removeValue: function(id) {
      return $http.delete('/api/statistics/' + id);
    }
  }
});
```

Esimerkkikoodi 5. Esimerkkisovellusta varten toteutettu AngularJS-palvelu.

Ajax-pyyntöjä varten toteutetun statisticsService-palvelun toteuttamiseen on käytetty AngularJS:n Factory-tyyppiä, joka on yksi AngularJS:n viidestä palvelutyypistä. Muita palvelutyyppejä ovat Value, Service, Provider ja Constant, joista jokaiselle on oma käyttötarkoituksensa. Factory-tyyppi soveltuu kuitenkin parhaiten statisticsService-palvelun toteuttamiseen, koska se mahdollistaa mukautettujen funktioiden tuomisen AngularJS-käsittelijöihin parhaiten. [45.]

Toteutettu statisticsService-palvelu saa parametrina AngularJS:n riippuvuusinjektio-ominaisuutta käyttäen AngularJS:n oman $http-palvelun, joka mahdollistaa Ajax-pyyntöjen tekemisen palvelimelle. Esimerkkikoodissa 5 on nähtävissä, että getData-, addValue- ja removeValue-funktioissa käytetään $http-palvelun get-, post- ja delete-funktioita, jotka vastaavat HTTP-protokollan samannimisiä pyyntöjä. [44.] Reitit on to-

teutettu vastaamaan aiemmin Laravelin reittienmääritystiedostoon määritettyjä reitityssääntöjä. Esimerkkikoodista 6 nähdään, miten mm. toteutetun statisticsService-palvelun toteutukset otetaan käyttöön AngularJS:n käsittelijässä.

```
app.controller('StatisticsCtrl',
               function($scope, statisticsService, statisticsData) {

  $scope.statisticsData = statisticsData;

  $scope.addValue = function() {
    statisticsService.addValue($scope.value).
      success(function(statisticsData) {
        $scope.statisticsData = statisticsData;
      }).error(function() {
        // handle error
      });
  };

  $scope.removeValue = function(id) {
    statisticsService.removeValue(id).
      success(function(statisticsData) {
        $scope.statisticsData = statisticsData;
      }).error(function() {
        // handle error
      });
  };

});
```

Esimerkkikoodi 6. Esimerkkisovellusta varten toteutettu AngularJS-käsittelijä.

Käsittelijä luodaan käyttäen luodun app-moduulimuuttujan controller-funktiota. Luotavalle käsittelijälle annetaan nimeksi StatisticsCtrl ja se saa parametrina AngularJS:n $scope-palvelun, aiemmin toteutetun statisticsService-palvelun ja sovelluksen konfiguraatioissa resolve-sanalla määritetyn statisticsData-objektin, joka sisältää palvelimelta haetun datan eli käyttäjän aiemmin syöttämät luvut ja lasketut tilastoluvut. [46.] AngularJS:n $scope-palvelulla voidaan viitata sovelluksen dokumenttimallin tiettyihin hierarkisen rakenteen osiin. Sen avulla käsittelijä ja näkymä saadaan liimattua yhteen, jolloin käsittelijässä tehtyjä mallimuutoksia saadaan suoraan näkyviin näkymään ja päinvastoin. [47.]

Palvelimelta haetut tiedot saadaan näkymän käyttöön asettamalla ne $scope-palvelun ominaisuudeksi aivan, kuten alustettaisiin muuttuja. Samalla tavalla $scope-palvelua käyttäen saadaan näkymässä laukaistun tapahtuman kautta kutsuttua $scope-palveluun asetettuja funktioita, joita ovat esimerkkikoodin 6 tapauksessa addValue- ja removeValue-funktiot, joissa kutsutaan toteutetun statisticsService-palvelun samanni-

misiä funktioita. Funktiot palauttavat $http-palvelun käytön vuoksi HttpPromise-objektin, jonka success-funktiota kutsutaan Ajax-pyynnön onnistuessa, jolloin tiedot päivitetään $scope-palvelua käyttäen näkymään. Vastaavasti Ajax-pyynnön epäonnistumista varten voidaan HttpPromise-objektin error-funktiota käyttämällä määrittää ohjeet virheenkäsittelyyn. Käytetyt success- ja error-funktiot eroavat hieman aiemmin AngularJS-sovelluksen konfiguroinnin yhteydessä käytetystä then-funktiosta mm. niille annettujen funktioiden saamien parametrien perusteella. [44; 47.]

Käsittelijässä $scope-palveluun asetetut ominaisuudet voidaan näyttää suoraan reitille määritetyssä näkymässä. Esimerkkikoodissa 7 on esitetty, miten palvelimella lasketut tilastoluvut voitaisiin asettaa osaksi käytettävän osasivun näkymää. Lasketut arvot esitetään yksinkertaisesti HTML:n kappaleteksteille tarkoitettujen p-tagien sisällä. AngularJS mahdollistaa $scope-palvelun ominaisuudeksi tallennetun datan näyttämisen näkymässä aaltosulkeita käyttämällä esimerkkikoodin 7 mukaisesti. [47.]

```
<p>Mean: {{ statisticsData.calculated.mean | round : 2 }}</p>
<p>Median: {{ statisticsData.calculated.median | round : 2 }}</p>
<p>Mode: {{ statisticsData.calculated.mode }}</p>
<p>Range: {{ statisticsData.calculated.range }}</p>
```

Esimerkkikoodi 7. Osasivussa sijaitseva lasketut tilastoluvut näyttävä koodi.

Esimerkkikoodissa 7 esiintyy lasketun keskiarvon ja mediaanin jälkeen putkimerkki ja putkimerkin jälkeen sana round, kaksoispiste ja luku 2. Tässä käytetään AngularJS-suodatinta, joka on toteutettu esimerkkiä varten itse (esimerkkikoodi 8). Se on tarkoitettu lukujen pyöristämistä varten kaksoispisteen jälkeen annetulla desimaalitarkkuudella. Toteutettavan tilastolukuja laskevan esimerkkisovelluksen tapauksessa vain keskiarvon ja mediaanin tuloksen on mahdollista olla desimaaliluku, koska käyttäjä voi syöttää sovellukseen vain kokonaislukuja.

```
app.filter('round', function() {
  return function(value, accuracy) {
    if (typeof value == 'number') {
      return value % 1 != 0 ? value.toFixed(accuracy) : value;
    }
    return value;
  };
});
```

Esimerkkikoodi 8. Esimerkkisovellusta varten toteutettu AngularJS-suodatin.

Suodatin luodaan käyttäen luodun app-moduulimuuttujan filter-funktiota. Esimerkkikoodin 8 tapauksessa suodattimelle annetaan nimeksi round ja sen saama value-parametri on suodattimen käytön yhteydessä ennen putkimerkkiä annettu luku ja accuracy-parametri on kaksoispisteen jälkeen annettu pyöristykseen käytettävä desimaalitarkkuus. [48.] Suodattimen toiminta on toteutettu niin, että jos suodattimen saama value-parametri on JavaScriptin number-tyyppinen liukuluku, pyöristetään luku käyttäen JavaScriptin omaa toFixed-funktiota. Suodatin ei pyöristä kokonaislukuja. Jos value-parametri ei ole number-tyyppinen, palautetaan se takaisin sellaisenaan.

Uuden luvun lisäämiseen tilastolukujen laskemista varten on tarkoitus käyttää toteutetun statisticsService-palvelun addValue-funktiota. Esimerkkikoodissa 9 on esitetty luvun lisäämistä varten toteutettu lomake, jonka sisältämää painonappia painamalla syöttökenttään syötetty luku voidaan lisätä palvelimella sijaitsevaan säilöön Ajax-pyynnön avulla.

```
<form ng-submit="addValue()">
  <add-value-input></add-value-input>
  <button>Add value</button>
</form>
```

Esimerkkikoodi 9. Osasivussa sijaitseva esimerkkisovellusta varten toteutettu lomake.

Lomakkeen form-tagissa käytetään AngularJS:n ngSubmit-direktiiviä. Tämä saa aikaiseksi sen, että painettaessa luvun lisäämiseen tarkoitettua nappia tarkistetaan ensin, onko syöttökentän sisältö sallittu, jonka jälkeen sallitun luvun lisäämiseen käytetään $scope-palvelulle asetettua addValue-funktiota. [49.] Tallennettava arvo saadaan syöttökentästä, joka on esimerkkikoodin 9 tapauksessa määritetty itse toteutetussa addValueInput-direktiivissä (esimerkkikoodi 10). AngularJS-direktiivit mahdollistavat ikään kuin omien HTML-elementtien luomisen. [50.]

```
app.directive('addValueInput', function() {
  return {
    restrict: 'AE',
    template: '<input type="number" min="0" max="99"' +
              'required ng-model="value"></input>'
  }
});
```

Esimerkkikoodi 10. Esimerkkisovellusta varten toteutettu AngularJS-direktiivi.

Direktiivi luodaan käyttäen luodun app-moduulimuuttujan directive-funktiota. Esimerkkikoodissa 10 luodulle direktiiville annetaan nimeksi addValueInput, jolloin sitä voidaan

käyttää esimerkkikoodissa 9 näkyvällä tavalla. Direktiivin sisällä määritetty restrict-asetus ja sille asetettu AE-arvo rajoittavat direktiivin käytön HTML:n attribuuttien ja elementtien käytön kaltaiseksi. C-arvo sallisi direktiivin käytön myös HTML:n class-attribuutin avulla. [50.]

Direktiivin sisällä on määritetty template-sanaa käyttäen, mitä direktiivin halutaan tuottavan. Tässä tapauksessa se tuottaisi input-elementin, jonka tyyppi on luku ja luvulle sallittu arvo olisi kokonaisluku väliltä 0–99. Lisäksi required-attribuutti määrittää sen, että syöttökenttä ei saa olla tyhjä luvun lisäämisnappia painaessa. Lopussa on vielä käytetty AngularJS:n ngModel-direktiiviä, joka asettaa syöttökentässä olevan arvon $scope-palvelun value-nimiseen ominaisuuteen, jolloin sitä voidaan käyttää käsittelijässä sellaisenaan, eikä syötettävää lukua ole tarvetta antaa argumenttina luvun lisäämiseen tarkoitetulle addValue-funktiolle. [50; 51.]

Sivulle tulisi luoda jokaista lisättyä lukua varten poistamisnappi, jota painamalla yksittäinen lisätty luku voitaisiin poistaa. Esimerkkikoodissa 11 on esitetty, miten kaikki luvut saadaan esille näkymään yksinkertaisena tekstinä ja miten jokaisen luvun viereen saadaan luotua poistamisnappi AngularJS:n ngRepeat-direktiivin avulla. Jos luvuista ja poistamisnapeista tehtävä taulukko haluttaisiin kuvan 5 rautalankamallin näköiseksi, pitäisi esimerkkikoodin 11 HTML-elementeille lisätä CSS-tyyliohjeita.

```
<div ng-repeat="(id, value) in statisticsData.values track by $index">
  <div>{{ value }}</div>
  <button ng-click="removeValue(id)">Delete</button>
</div>
```

Esimerkkikoodi 11. Osasivussa sijaitseva tallennetut luvut näyttävä ja poistonapit luova koodi.

Lukutaulukkoa edustavan div-tagin sisällä käytettävälle ngRepeat-direktiiville ilmoitetaan, että yksittäisen taulukon alkion indeksi halutaan käyttöön nimellä id ja itse alkion arvo halutaan käyttöön nimellä value. Taulukko, joka iteroidaan, otettiin käyttöön asettamalla se objektina $scope-palvelun statisticsData-nimiseen ominaisuuteen. Itse iteroitavat arvot ovat kyseisen objektin sisällä nimellä values. Käytettävälle ngRepeat-direktiiville joudutaan lisäksi kertomaan, että iteroitavan taulukon lukuja halutaan jäljittää niiden indeksien perusteella, koska taulukko voi sisältää useita samoja lukuja. Luvun poistamisnappia painaessa voidaan kutsua ngClick-direktiivin avulla luvun poistamiseen tarkoitettua removeValue-funktiota, jolle annetaan argumenttina poistettavan luvun tunnus. [52; 53.]

Lopulta kaikki edellisten AngularJS-esimerkkien toiminnallisuus saadaan käyttöön käyttäen ngApp-direktiiviä sovelluksen päädokumentissa. Lisäksi näytettävien osasivujen sisältö saadaan näkyviin ngRoute-moduulin ngView-direktiiviä käyttäen, joka näyttää käytetyn reitin mukaisen osasivun. Esimerkkikoodissa 12 on esitetty, miten toteutettu AngularJS-sovellus saadaan käyttöön sovelluksen päädokumentissa sijaitsevan div-elementin sisällä. Sisältöä voidaan myös lisätä div-elementin ulko- ja sisäpuolelle, jolloin sisältö näytettäisiin reitistä huolimatta. Lisäksi kirjoitetut AngularJS-skriptit pitää ottaa erikseen käyttöön, jotta sovellus toimisi. [54; 55.]

```
<div ng-app="statisticsCalculator">
  <ng-view></ng-view>
</div>
```

Esimerkkikoodi 12. AngularJS-sovelluksen käyttöönotto sovelluksen päädokumentin sisällä.

Edellisissä AngularJS-esimerkeissä esitettyjä ominaisuuksia käyttäen insinöörityönä kehitettävää laskentasovellusta varten saadaan helposti toteutettua vaadittua toiminnallisuutta sovelluksen asiakaspuolelle. Sovelluksen käyttäjien on mahdollista lukea kaikkia edellisiä AngularJS-esimerkkikoodeja, koska käyttäjien selaimilla on oltava pääsy niihin. Insinöörityönä kehitettävän laskentasovelluksen vaatimuksissa oli määritetty, ettei käyttäjillä saisi olla pääsyä sovelluksen laskentalogiikkaan. Edellisissä AngularJS-esimerkeissäkään ei ole nähtävissä tilastolukuja laskevan laskentalogiikan toteutustatapaa, koska laskutoimitukset suoritetaan palvelimella. Sovelluksen käyttämien Ajax-pyyntöjen ansiosta sivua ei tarvitse uudelleenladata laskuja tehdessä koskaan, joten sovelluksen käyttökokemuksesta saadaan sujuva.

Käyttöliittymän ulkoasu on myös oleellinen osa asiakaspuolta. Kuten aiemmin on jo tullutkin ilmi, sisälsi insinöörityönä kehitettävä laskentasovellus paljon toivottuja ominaisuuksia ja tietynlainen käyttövarmuus oli vaadittua. Sovelluskehitykseen käytettävän ajan puutteen ja suuren työmäärän vuoksi tilaajan kanssa sovittiin jo projektin alkuvaiheilla, että tehtävässä sovellusprototyypissä ei juurikaan puututtaisi sovelluksen ulkoasuun liittyviin asioihin.

4.3 Tietojen tallennusmenetelmät

Insinöörityönä kehitettävän laskentasovelluksen täytyy pystyä tallentamaan käyttäjätietoja sekä käyttäjien työstämiä projekteja pysyvästi. Laravel tukee tietokantatallennusta

MySQL-, PostgreSQL-, SQLite- ja Microsoft SQL Server -tietokantoihin suoraan. Sovelluskehityksessä ei ole käytännössä merkitystä, mitä näistä käytetään, sillä Laravel mahdollistaa käytettävän tietokannan vaihtamisen todella helposti ilman hankalaa konfigurointia. [56.] Ennen sovelluksen julkaisemista tulisi kuitenkin miettiä tietokantavaihtoehtojen eroja ja soveltuvuutta käyttötarkoitukseen tarkemmin.

Laravel tarjoaa valmiin Eloquent ORM -ratkaisun (Object Relational Mapper) tietokantaan tallennettavien mallien käsittelyyn. Jokainen Eloquent-luokasta periytyvä luokka vastaa sille tarkoitettua tietokantataulua sovelluksen tietokannassa ja luokille voidaan tehdä tietokantaoperaatioita Eloquent-luokan valmisfunktioiden avulla, jolloin erillisiä tietokantakyselyjä ei tarvitse kirjoittaa itse lainkaan. [57.] Eloquent-luokkaa tullaan käyttämään insinöörityönä kehitettävän laskentasovelluksen tapauksessa käyttäjien ja projektien tietokantatallennukseen. Toteutukseen käytetään kuitenkin säilömallia, joten tarvittaessa vaihto toiseen ratkaisuun on helpompaa. Käyttäjätilit ja projektit ovat laskentasovelluksen tapauksessa ainoat tietokantaan tallennettavat tiedot.

Suunnitteluvaiheessa tuli jo ilmi, että sovelluksessa työstettävän aktiivisen projektin tietoja ei haluta tallentaa pysyvästi mihinkään, joten pysyvää tietokantatallennusta ei pidetty edes vaihtoehtona. Tiedot täytyisi kuitenkin tallentaa johonkin tietovarastoon väliaikaisesti, josta ne voitaisiin hakea projektin työstämisen aikana. Tietovaraston tulisi tyhjentyä käyttäjän kirjautuessa ulos sovelluksesta tai tietyn käyttäjän toimettomuusajan jälkeen. Istuntotallennuksen todettiin soveltuvan tähän käyttötarkoitukseen. Laravel tukee istuntotallennusta tiedostoihin, salattuihin evästeisiin, tietokantaan ja Memcached- ja Redis-palvelimien avulla myös muistiin. Laravel mahdollistaa myös istuntotallennustavan vaihtamisen helposti ilman hankalaa konfigurointia, joten vasta sovelluksen käyttöönoton yhteydessä tulisi käyttötarkoitukseen valita sopivin vaihtoehto. [58.]

Istuntoon tallennetut käyttäjäkohtaiset tiedot pysyvät istunnossa niin kauan kuin käyttäjän istunto on voimassa eli istuntotietoihin ei ole enää pääsyä istunnon vanhenemisen jälkeen. Näin ollen työstettävän projektin dataa ei tallenneta mihinkään pysyvästi. [58.] Taas olisi kuitenkin hyvä käyttää säilömallia tallennusmenetelmän toteutuksessa, koska jos jostakin syystä myöhemmin todetaan, ettei toteutettu istuntotallennus sovellukaan käyttötarkoitukseen, voitaisiin työstettävien projektien tallennusmenetelmä muuttaa mahdollisimman helposti esimerkiksi pysyväksi tietokantatallennukseksi. Käsittelijäluokkiin ei siis taaskaan tarvitsisi tehdä muutoksia tallennustavan muuttuessa.

Projektien tallennuksen pitää onnistua tietokantatallennuksen lisäksi myös käyttäjien omille työasemille. Käyttäjien olisi tällöin mahdollista tuoda projekti takaisin sovellukseen lähettämällä projektitiedosto palvelimelle, josta palvelin lukee projektin tiedot ja tallentaa ne istuntoon. Varteenotettavimmat tiedostoformaatit projektitiedostoille ovat JSON ja XML (Extensible Markup Language). XML-formaatti esittää kuitenkin suuria lukujoukkoja tehottomammin sisällyttämällä esitysmuotoon turhaa elementtien vaatimaa tekstiä, joten JSON-formaatti soveltuu näistä vaihtoehdoista paremmin vaadittuun käyttötarkoitukseen selkeämmän rakenteen vuoksi. [59.]

Oletetaan, että aiemmin toteutetun tilastolukuja laskevan yksinkertaisen laskentasovelluksen tulisi olla mahdollista tallentaa käyttäjien syöttämiä lukuja tiedostoihin. Esimerkkikoodissa 13 on esitetty, miltä JSON-tiedoston sisältö voisi näyttää, jos käyttäjä olisi syöttänyt lukuja yhdestä kymmeneen. Kyseisen esimerkin tapauksessa JSON-tiedostossa on vain yksi ominaisuus nimeltään values, joten käytännössä sisältö voitaisiin esittää myös yhdessä pelkästään lukuja sisältävässä nimettömässä JSON-taulukkotyypissä. [26.] Varsinaisen insinöörityönä kehitettävän laskentasovelluksen tapauksessa projektitiedostot sisältäisivät lukujen lisäksi kuitenkin useita muitakin ominaisuuksia, kuten erilaisia laskentaa ohjaavia tietoja, joten esimerkinkin tapauksessa lukutaulukko on nimetty, jolloin sen rinnalle voitaisiin lisätä muitakin tietoja. Esimerkin luvut eivät sisällä yksilöiviä tunnuksia, mutta niille voidaan luoda tunnukset niiden järjestysnumeroiden perusteella, jolloin lukuja olisi mahdollista poistaa aiemmin toteutetun laskentasovellusesimerkin tapauksessa luotujen tunnusten perusteella.

```
{
  "values": [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
}
```

Esimerkkikoodi 13. Esimerkki JSON-tiedoston sisällöstä.

JSON-tiedostojen sisältöjen validointi on mahdollista JSON Schema -spesifikaation avulla, jota varten löytyy useita ohjelmointikieliä varten tehtyjä toteutuksia. JSON Schema on eräänlainen joukko sääntöjä JSON-tiedostomallin määrittämiseksi. Tiedostomallia määrittäessä voidaan JSON-tiedostossa oleville tietokentille määrittää tietotyypit, sallittavat sisällöt ja tietokenttien pakollisuus. Määritettyä mallia voidaan sitten verrata JSON-tiedostoihin ohjelmointikielikohtaisella työkalulla. [60.] Esimerkkikoodiin 14 on määritetty JSON Schema -spesifikaatiota noudattavat säännöt esimerkkikoodissa 13 esiintyvän JSON-tiedoston sisällön validoimista varten.

```
{
  "id": "http://statistics-calculator.com/statistic-values-
schema.json#",
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "Statistic values",
  "description": "The values entered by the user to calculate
statistics",
  "type": "object",
  "properties": {
    "values": {
      "type": "array",
      "items": {
        "description": "An individual value",
        "type": "integer",
        "minimum": 0,
        "maximum": 99
      }
    }
  },
  "required": [
    "values"
  ],
  "additionalProperties": false
}
```

Esimerkkikoodi 14. JSON Schema -spesifikaatiota noudattavat säännöt esimerkkikoodin 13 mukaisen JSON-tiedoston sisällölle.

Esimerkkikoodin 14 ensimmäinen ominaisuus on id, joka määrittää kyseisen JSON Schema -tiedoston resurssipolun, joka on esimerkin tapauksessa määritetty vastaamaan olematonta polkua. Toinen ominaisuus on $schema, joka määrittää JSON Schema -spesifikaation version, jota noudattaen kyseinen mallitiedosto on kirjoitettu. Ominaisuudet title ja description ovat pelkästään mallin sisältöä kuvaavia ominaisuuksia. Ominaisuus type määrittää kuvailtavan mallin tietotyypin ja properties määrittää object-tietotyypin kohdalla sen sisällöt. Malliin on määritetty, että object-tyyppi voi sisältää values-nimisen ominaisuuden, jonka on oltava array-tietotyyppinen ja se saa sisältää vain integer-tyyppisiä kokonaislukuja, joiden on oltava väliltä 0–99. Lopulta required-ominaisuuteen on määritetty, että JSON-tiedoston on sisällettävä values-niminen ominaisuus ja additionalProperties-ominaisuudelle annettu false-arvo määrittää, ettei JSON-tiedosto saa sisältää values-ominaisuuden lisäksi muita ominaisuuksia. [61; 62.]

Insinöörityönä kehitettävän laskentasovelluksen tapauksessa pelkkä projektitiedoston sisällön vertaaminen määritettyyn JSON Schema -malliin ei kuitenkaan riitä, koska osa projektitiedoston tietokentistä pakottaa toiset tietokentät sisältämään joitakin tiettyjä arvoja. Tämän vuoksi koodiin joudutaan kirjoittamaan JSON Schema -mallin validoin-

nin lisäksi manuaalisia tarkistuksia. Manuaaliset tarkistukset tullaan tekemään palvelinpuolella PHP-ohjelmointikielen ehtorakenteita käyttäen.

4.4 PDF-raporttien koostaminen

PDF-raportit ovat sovelluksen luomien raporttien osalta haasteellisimpia. Kaikkien raporttien laskentalogiikka toimii samalla tavalla riippumatta käyttäjän valitsemasta raporttityypistä. Selaimella HTML-muodossa näytettävät raportit palautetaan JSON-muodossa palvelimelta selaimelle vastauksena Ajax-pyyntöön. CSV-raportit ovat käytännössä vain muotoiltuja merkkijonoja, jotka kirjoitetaan palvelimelta ladattaviin tiedostoihin. PDF-raporttien luomiseen tarvitaan sen sijaan erillistä työkalua.

Kaikille raporteille jouduttiin tekemään HTML-osasivut, koska raportit tuli pystyä näyttämään selaimella HTML-muodossa. Tästä syystä PDF-raporttien luomisessa päädyttiin ratkaisuun, jossa PDF-tiedostojen sisältö luodaan HTML-merkkauskieltä käyttäen. Osasivuja joudutaan kuitenkin hieman muokkaamaan, koska selaimella näytettävät sivut sisältävät muutakin merkkausta ja tiedot tulee saada mahtumaan PDF-tiedostojen tapauksessa A4-kokoon.

PDF-tiedostoihin joudutaan piirtämään näkymä jo palvelinpuolella, koska tiedostot tulee luoda jo palvelinpuolella latausta varten. Laravel tarjoaa yksinkertaisen ja tehokkaan Blade-työkalun HTML-sivujen luomiseen. Blade mahdollistaa PHP-skripteissä määritettyjen muuttujien, ohjaus- ja silmukkarakenteiden ja muiden osasivumallien käytön samassa näkymässä. [63.] Esimerkkikoodissa 15 on esimerkki ehto- ja silmukkarakenteen sekä muuttujien käytöstä Blade-näkymässä.

```
@if (empty($values))
  Array does not contain any values.
@else
  <table>
    <tr>
      @for ($i = 0; $i < count($values); $i++)
        <td>{{ $values[$i] }}</td>
      @endfor
    </tr>
  </table>
@endif
```

Esimerkkikoodi 15. Laravelin Blade-näkymä, jossa on käytetty muuttujia, ehtorakennetta ja silmukkarakennetta.

Blade-näkymä sisältää HTML-merkkausta, johon on lisätty Blade-työkalun omaa syntaksia. Esimerkkikoodin 15 alussa käytetään if-ehtorakennetta, jossa tarkistetaan Blade-näkymälle annetun taulukkomuuttujan sisältö. Jos taulukossa ei ole alkioita, näytetään näkymässä asiasta ilmoittava teksti. Muussa tapauksessa taulukko iteroidaan forsilmukassa, jolloin taulukon sisältö saadaan näkyviin HTML:n taulukkoelementtejä käyttäen. Muuttujien arvot saadaan esille näkymään aaltosulkeita käyttämällä esimerkkikoodin 15 mukaisesti. [63.] Esimerkki on siltä osin yksinkertainen, että taulukon sisältö ulottuu A4-kokoisissa PDF-tiedostoissa A4-näkymän ulkopuolelle taulukon sisältäessä liikaa alkioita. Tiedon esittämistä varten pitäisi siis lisätä enemmän tietoa ohjaavaa logiikkaa.

HTML-merkkauksen muuntamiseen PDF-tiedostoksi löytyi useita eri työkaluvaihtoehtoja. Kaksi varteenotettavinta vaihtoehtoa olivat kuitenkin dompdf ja wkhtmltopdf, koska niitä varten löytyi Laravelia varten toteutetut vakaat ja julkaistavaan sovellukseen soveltuvat lisäosat. Molempien työkalujen asennus onnistui suht vaivattomasti ja molemmat saatiin tuottamaan ladattavia PDF-tiedostoja. Näistä dompdf oli kuitenkin huomattavasti hitaampi, jos näkymään piti saada esille suuria lukutaulukoita. Näin ollen PDF-raportit tullaan toteuttamaan wkhtmltopdf-työkalulla ja sitä käyttävän Laravel Snappy PDF -lisäosan avulla. Esimerkkikoodissa 16 on esitetty, kuinka Laravel Snappy PDF -lisäosaa käyttämällä saadaan luotua PDF-tiedosto ja ladattua se työasemalle. [64; 65.]

```
$snappy = App::make('snappy.pdf.wrapper');
$values = array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
$pdf = $snappy->loadView('example', array('values' => $values));

$pdf->setPaper('A4')->setOrientation('landscape');

return $pdf->download('example.pdf');
```

Esimerkkikoodi 16. PDF-tiedoston luominen ja lataaminen työasemalle Laravel Snappy PDF -lisäosan avulla.

Esimerkkikoodin 16 alussa luodaan Laravel Snappy PDF -lisäosan PdfWrapper-luokka käyttäen Laravelin App-luokkaa, jonka avulla haluttu luokka saadaan käyttöön Laravelin käänteisen hallinnan säiliön avulla ilman luokan alustamista new-sanalla. Seuraavaksi alustetaan esimerkkitietoja taulukkomuuttujaan. Samaa taulukkomuuttujan nimeä käytettiin myös esimerkkikoodissa 15, jossa taulukko iteroitiin Laravelin Blade-näkymässä. Tämän jälkeen luodaan pdf-muuttuja käyttäen PdfWrapper-luokan loadView-funktiota, jolle annetaan argumentteina käytettävän näkymämallin nimi ja näkymässä käytettävät muuttujat array-tietotyypin sisällä. Näkymämallin nimi on esimerkin

tapauksessa example ja näkymälle annetaan muuttujana lukuja sisältävä taulukkomuuttuja. Seuraavaksi setPaper-funktiolla asetetaan PDF-tiedoston kooksi A4-koko ja setOrientation-funktiolla käännetään PDF-tiedoston näkymä sivuttain. Lopuksi kutsutaan luodun muuttujan download-funktiota, jonka avulla tiedosto saadaan ladattua omalle työasemalle download-funktiolle annetun argumentin nimisenä. [65.]

Sovellukseen saatiin toteutettua ominaisuudet PDF-raporttien luomista varten, mutta raporttien ulkoasuun ei tässäkään tapauksessa kiinnitetty vielä suurempaa huomiota. Raporttien ulkoasuja pystytään muokkaamaan enemmän ajan salliessa tyylittelemällä raporttien luomiseen käytettäviä näkymiä.

4.5 Työn dokumentointi

Koska olin ainoa sovelluksen kehittäjä, toivottiin sovelluksesta jonkinlaista dokumentointia sen varalle, että jos jatkossa en olekaan enää mukana sovelluksen kehitystyössä ja tilalle tuleekin muita kehittäjiä, saisivat he mahdollisimman helposti ymmärryksen tekemieni ratkaisujen toimintaperiaatteista. Varsinaista sovelluksen toimintaan liittyvää lisäarvoa dokumentointi ei toisi, mutta se auttaisi sovelluksen jatkokehityksessä ja muutosten tekemisessä, joten dokumentointi haluttiin ottaa osaksi itse sovelluskehitysprosessia.

On kuitenkin mainitsemisen arvoista, että kattavan dokumentoinnin tärkeyttä pidetään ohjelmistokehitysmaailmassa ehkä jopa hieman kyseenalaisena. Ketterän ohjelmistokehityksen arvoihin kuuluu, että toimiva ohjelmisto on tärkeämpää kuin kattava dokumentaatio ja koodista pitäisi tehdä jo itsessään mahdollisimman ymmärrettävää käyttäen selkeitä nimeämissääntöjä ja toimintaperiaatteita. [66.] Tehtävästä dokumentaatiosta ei tulla tekemään liian kattavaa, vaan lähinnä ohjeistava.

Sovelluksen palvelinpuolen PHP-koodit dokumentoidaan phpDocumentor-työkalun avulla sen luoman dokumentoinnin siisteyden ja selkeyden perusteella. Se luo dokumentointia koodiin kirjoitetuista DocBlock-kommenteista, joiden tarkoitus on kuvata yksittäisten luokkien, funktioiden ja rakenteellisten elementtien toimintaa. Työkalun luomaa dokumentointia on tarkoitus lukea selaimella ja dokumentoinnin ulkoasua varten on käytettävissä useita eri mallipohjia. Dokumentoinnin lukeminen ei myöskään

vaadi käynnissä olevaa palvelinta, koska työkalun luoman dokumentoinnin resurssit ovat puhtaasti selaimella suoritettavia. [67.]

Liite 1 sisältää phpDocumentor-työkalulla luotua esimerkkidokumentointia, joka on luotu käyttämällä työkalun oletusmallia. Dokumentointi on tehty aiemmin esimerkkinä toteutetun tilastolukuja laskevan sovelluksen palvelinpuolta varten. Liitteen 1 ensimmäisessä kuvassa on esitetty luodun dokumentoinnin yleisnäkymää, jossa pystytään navigoimaan eri dokumentoitujen komponenttien välillä. Toisessa kuvassa on esitetty yksittäisen luokan tarkastelunäkymä dokumentoinnissa. Näkymä sisältää kaikki kyseistä luokkaa varten kommentoidut funktiot ja muuttujat. Kolmannessa kuvassa on näkymä, jonka avulla voidaan lukea yksittäisen dokumentoidun tiedoston lähdekoodeja. Viimeisessä kuvassa on vielä esitetty työkalun luoma yksinkertaistettu luokkakaavio, joka kuvaa dokumentoitujen luokkien ja rajapintojen hierarkiaa sovelluksessa.

Sovelluksen asiakaspuolen JavaScript-koodit dokumentoidaan Angular-JSDoc-työkalun avulla, joka on lähinnä yksittäisen henkilön kehittämä yksinkertainen JSDoc-työkalun varaan rakennettu liitännäinen, joka mahdollistaa myös AngularJS:lle ominaisten komponenttien dokumentoinnin. Angular-JSDoc ei luo läheskään yhtä siistiä dokumentointia kuin phpDocumentor, mutta koettiin käyttötarkoitukseen täysin riittäväksi. Sekin luo selaimella luettavaa dokumentointia koodiin kirjoitetuista kommenteista, eikä dokumentoinnin lukemiseen tarvita käynnissä olevaa palvelinta. [68.]

Liite 2 sisältää esimerkkejä Angular-JSDoc-työkalulla tehdystä dokumentoinnista. Dokumentointi on tehty aiemmin esimerkkinä toteutetun tilastolukuja laskevan sovelluksen asiakaspuolta varten. Ensimmäisessä kuvassa näkyy yksittäisen AngularJS-komponentin tarkastelu. Dokumentointi näyttää funktioiden parametrit ja palautettavat tyypit. Kuvan oikeassa reunassa on navigointipalkki, joka sisältää linkit kaikista komponenteista tehtyyn dokumentointiin. Kaikki erityyppiset AngularJS-komponentit ovat omien otsikoiden alla. Toisessa liitteen 2 kuvassa on nähtävissä yksittäisen komponentin lähdekoodien tarkastelu suoraan selaimella. Samassa kuvassa on nähtävissä myös lähdekoodin kommenttilohkoja, joista dokumentointi on luotu.

5 Jatkokehitys

5.1 Lisäominaisuudet

Sovellukseen on suunniteltu toteutettujen ominaisuuksien lisäksi muutamia lisäominaisuuksia, joiden toteuttamiseen ei ollut tarpeeksi aikaa. Tarkoituksena on, että sovellusta voitaisiin käyttää pelkästään aktiivisella käyttäjälisenssillä, jolloin sovelluksessa olisi tietyn ajan kestävä tilausmahdollisuus. Käyttäjätileihin pitäisi siis lisätä käyttäjäkohtaisia lisenssitietoja, jotka sallivat sovelluksen käytön tiettyyn päivämäärään asti. Toistaiseksi on suunniteltu, ettei tilausominaisuutta vielä automatisoitaisi mitenkään. Asiakkaat olisivat ainakin aluksi erikseen yhteydessä palveluntarjoajaan tilausta halutessaan.

Laravel tarjoaa joka tapauksessa oman ratkaisunsa tilauksen automatisoinnin toteuttamiseen. Laravel sisältää oman Cashier-toteutuksensa, joka käyttää Stripe-maksujärjestelmää. Cashierin avulla Laravelin käyttäjämallien tietokantatauluun saadaan helposti lisättyä tietoja käyttäjien lisensseihin liittyen ja sovellukseen on mahdollista lisätä myös tietyn ajan kestävä kokeilumahdollisuus. Stripe mahdollistaa tilaamisen kaikilla tavanomaisilla maksumahdollisuuksilla, kuten suosituimmilla luottokorteilla. Tilauksen kestolle on myös mahdollista määrittää useita vaihtoehtoja. Jokaista tilausta kohden Stripe veloittaa kuitenkin pienen summan palvelumaksuna. Todennäköisesti maksun automatisointiin halutaan kuitenkin tulevaisuudessa käyttää paremmin suomalaisten pankkien tilisiirtoja tukevaa maksujärjestelmää. Tätä varten joudutaan kuitenkin päivittämään sovelluksen käyttäjämallien rakennetta lisenssejä varten suuremmalla työllä verrattuna Laravelin Cashierin käyttöön. [69; 70.]

Sovellukseen halutaan myös omat sovelluksen sisäiset ylläpitosivut, joiden avulla insinöörityön tilaaja pystyisi hallinnoimaan käyttäjätilejä mahdollisimman helposti myös itse. Tilaaja voisi hallintosivujen avulla lisätä itse uusia käyttäjätilejä sovelluksen tietokantaan ja myös poistaa niitä. Asiakas voi esimerkiksi vaatia oman käyttäjätilinsä poistamista järjestelmästä, jolloin tilaaja voisi tehdä sen itse ylläpitosivujen avulla. Käyttäjätilien poistamisen yhteydessä voitaisiin poistaa kaikki käyttäjätilille kuuluvat projektit tietokannasta. Toistaiseksi halutaan, että tämän kaltaiset käyttäjäkohtaiset toimenpiteet toteutetaan järjestelmän ylläpitäjän kautta, mutta yksinkertaisia toimenpiteitä, kuten unohtunutta salasanaa varten voitaisiin kuitenkin toteuttaa automatisoitu palautusmahdollisuus.

Käyttäjärakennetta halutaan myös mahdollisesti laajentaa tulevaisuudessa tukemaan organisaatiorakenteita. Käyttäjät voisivat siis kuulua johonkin organisaatioon, ja organisaation sisällä olisi mahdollista jakaa projekteja muiden käyttäjien kanssa. Organisaatioille pitäisi toteuttaa oma tietokantarakenne. Ajatuksena on myös alun perin ollut, että insinöörityönä kehitetty sovellus tulisi olemaan tulevaisuudessa vain yksi suurempaan kokonaisuuteen kuuluva sovellus. Yksittäisiin osasovelluksiin tulisi myös mahdollisesti pystyä ostamaan lisenssejä erikseen. Nämä laajennukset ovat tällä hetkellä vasta ajatusasteella, mutta sovellusta on kehitetty koko ajan ottaen huomioon nämä tulevaisuuden laajennusmahdollisuudet.

Sovelluksen käyttöliittymä jäi ajan puutteen ja useiden kehitettyjen ominaisuuksien vuoksi hyvin yksinkertaiseksi. Lisäksi sovelluksen luomat PDF-raportit eivät ole kovin siistejä. Näihin asioihin joudutaan myös panostamaan jatkossa, koska sovelluksen käyttökokemuksesta halutaan mahdollisimman miellyttävä.

5.2 Testaus

Sovellus sisältää osittain monimutkaistakin laskentalogiikkaa ja tietoja, jotka vaikuttavat muihin tietoihin. Lisäksi sovelluksen luonne on sellainen, että sen on toimittava varsinkin laskentalogiikan suhteen täysin virheettömästi, jotta se tuottaisi käyttäjilleen minkäänlaista arvoa. Virheellinen laskenta ajaisi käyttäjiä jopa ongelmiin, koska käyttäjät eivät itse tietäisi laskennan toimivan virheellisesti. Tämän vuoksi sovelluksen toiminnasta tulisi varmistua perusteellisesti ennen sen käyttöönottoa.

Sovelluksen toimintavarmuutta saadaan varmistettua kirjoittamalla sovelluksen toimintaa testaavia suoritettavia testejä. Testaus jäi työmäärän vuoksi valitettavan vähäiseksi ja sovellukseen tuli usein odottamattomia rakennemuutoksia, jolloin testitkin oltaisi jouduttu kirjoittamaan kokonaan uudestaan. Sovelluksen laskentalogiikka oli kuitenkin sellainen asia, jota varten testien kirjoittaminen oli lähes välttämätöntä. Tilaajan tekemästä Excel-työkirjaan sisäänrakennetusta työkaluversiosta saatiin kopioitua testien kirjoittamista varten testiarvoja, joita pystyttiin vertaamaan verkkosovelluksen laskentalogiikan tuottamiin lukuihin.

Testien kirjoittamiseen ja suorittamiseen käytettiin PHP-koodien automaattiseen testaukseen tarkoitettuja PHPUnit- ja Mockery-sovelluskehyksiä. Sovellukseen kehitetyt

laskentaluokat laskevat pääasiassa taulukkomaista dataa ja useimmiten laskentaan tarvitaan myös jo laskettuja arvoja. Täten jokaisen sovelluksen laskentalogiikan laskeman taulukon alkion tuli vastata ennalta määritettyä vastinettaan. PHPUnit mahdollistaa yksikkötestien kirjoittamisen ja suorittamisen, joiden avulla esimerkiksi yksittäisen olemassa olevan laskentaluokan tuottamia lukuja voidaan verrata ennaltamääritettyihin lukuihin. [71.]

Joissakin tapauksissa sovelluksen yhden laskentaluokan sisällä voi olla toinen laskentaluokka, jotta yksittäistä toiminnallisuutta on saatu pilkottua pienempiin osiin. Testauksen kannalta tässä tulee esille pieni haaste, koska laskentaluokan sisällä olevan laskentaluokan virheellinen toiminta saa myös ylemmän laskentaluokan testit epäonnistumaan. Testit pitäisi kirjoittaa aina vain yhdelle luokalle kerrallaan, jotta sovelluksen virheellinen toiminta saataisiin kohdistettua johonkin tiettyyn osaan. Näissä tapauksissa laskentaluokan sisällä ovelasta laskentaluokasta voidaan tehdä Mockery-sovelluskehyksen avulla mock-objekti, joka saadaan ikään kuin tekeytymään oikeaksi luokaksi. Testeissä voidaan tällöin määrittää itse, mitä tämän objektin tulisi tehdä, kun sitä yritetään käyttää. Näin ollen ylemmällä laskentaluokalla ei ole kirjoitetun testin suorituksessa riippuvuutta sen sisällä olevan laskentaluokan varsinaiseen toimintaan, joten jos testi ei läpäise, voidaan syy kohdistaa vain testattavaan luokkaan eikä ulkoisiin riippuvuuksiin. [72.]

Sovelluksen asiakaspuolta ei juurikaan testattu, mutta sekin on mahdollista omilla työkaluillaan. AngularJS-koodeja on mahdollista yksikkötestata käyttäen Karmaa ja Jasminea. Karma on komentorivityökalu, jonka avulla voidaan käynnistää testejä suorittava palvelin. Karmaan on mahdollista konfiguroida useita eri selaimia, joten se mahdollistaa myös sovelluksen selaintuen testaamisen. Jasmine on sovelluskehys JavaScript-koodien testaukseen ja siitä on tullut suosituin vaihtoehto AngularJS-sovelluksien testaamisessa. Jasmine sisältää tarvittavat toteutukset ja funktiot testien kirjoittamista varten. [73.]

Sovelluksen asiakaspuolen toiminnallisuutta voidaan myös simuloida automaattisesti niin, että ikään kuin oikea käyttäjä käyttäisi sovellusta selaimella. Tällöin puhutaan end-to-end-testauksesta. Testissä voidaan esimerkiksi käskeä käyttäjää painamaan jotakin sivulla olevaa nappia ja testata, tuottaako napin painaminen halutun lopputuloksen, joka voisi olla esimerkiksi siirtyminen toiselle sovelluksen sivulle. Vastaavanlainen testaaminen onnistuu Protractor-työkalulla, jonka avulla käyttäjän toimenpiteitä voidaan

simuloida ohjaamalla selaimen toimintaa. Myös Protractor käyttää Jasmine-sovelluskehystä testien kirjoittamisen apuna. [74.]

Testaamiseen on olemassa useita työkaluja ja testauksen hyödyt ovat selkeitä: koodista saadaan selkeämpää ja toiminnallisuus saadaan taattua. Testien kirjoittaminen vaatii kuitenkin sitä enemmän aikaa ja resursseja, mitä useampia toiminnallisuuksia testataan. Testaukseen täytyy osata suhtautua myös kriittisesti määrittämällä yksittäisten ominaisuuksien testauksen tärkeellisyys.

5.3 Tietoturvallisuus

Aukot sovellusten tietoturvassa antavat automatisoiduille boteille ja krakkereille eli tietojärjestelmään murtautuville henkilöille mahdollisuuden tartuttaa sovellukseen pahantahtoista koodia esimerkiksi roskapostin lähettämiseen tai tietokantadatan käsittelyyn. Identiteettivarkaudet ovat myös olennaisia uhkia ja sovelluksen käyttäjillä ei pitäisi olla lähtökohtaisesti pääsyä heiltä estetyksi tarkoitettuun toiminnallisuuteen. Sovelluskehitykseen käytetyt sovelluskehykset ottavat suurimpia tietoturvariskejä jo itsessään huomioon, mutta viimeistään sovelluksen käyttöönoton yhteydessä tietoturva-asiat tulisi ottaa tarkempaan tarkasteluun.

Maailmanlaajuinen ohjelmistojen turvallisuuteen keskittynyt voittoa tavoittelematon järjestö OWASP (The Open Web Application Security Project) listaa joka kolmas vuosi kymmenen suurinta verkkosovellusten tietoturvallisuuteen liittyvää uhkaa tietoisuuden herättämiseksi tietoturva-asioihin liittyen. Seuraavaksi listataan OWASPin kymmenen suurinta tietoturvauhkaa lyhyine selityksineen ja miten ne tulisi ottaa tai on mahdollisesti jo otettu kehitetyn laskentasovelluksen tapauksessa huomioon. [75.]

1. Injektio

Injektiolla tarkoitetaan epäluotettavien komentojen lähettämistä palvelimen suoritettavaksi esimerkiksi huonosti toteutetun tietokantakyselyn osaksi, jolloin hyökkääjä voi päästä käsiksi häneltä estetyksi tarkoitettuun dataan [75].

Kaikki tietokantaoperaatiot on toteutettu laskentasovellukseen Laravelin toteutuksia käyttäen, jotka puhdistavat kyselyjä varten annettuja epäluotettavia komentoja ennen kyselyjen suorittamista [76].

2. Käyttäjän todentamisen tai istunnon hallinnan toteuttaminen virheellisesti

Käyttäjän todentamisen tai istunnon hallinnan toteuttaminen virheellisesti saattaa antaa hyökkääjille useita mahdollisuuksia ottaa haltuun muiden käyttäjien käyttäjätunnuksia [75].

Käyttäjien salasanat salataan Laravelin Hash-luokan käyttämän salausalgoritmin avulla ennen salasanojen tallentamista tietokantaan. Istunnoille on määritetty vanhenemisaika, istuntojen tunnisteet vaihdetaan aina uudelleenkirjautuessa ja käyttäjien istuntotiedot nollataan uloskirjautuessa. Lisäksi istuntojen tunnisteet ovat pitkiä eikä niitä sisällytetä osoitepolkuihin. [77.] Sovelluksen käyttöönoton yhteydessä on vielä otettava käyttöön salausprotokollaa käyttävä yhteys.

3. Cross-Site Scripting (XSS)

XSS on haavoittuvaisuus, jota hyödyntäen hyökkääjä voi lähettää suoritettavia asiakaspuolen skriptejä toisten käyttäjien avaamille sivuille [75].

Sovelluksessa käytetään tiedon esittämiseen AngularJS:n toteutuksia, jotka estävät XSS-haavoittuvaisuuksia puhdistamalla epäluotettavaa sisältöä syöttötiedoista [78]. Kaikki palvelimelle lähetettävä data tulisi vielä puhdistaa ngSanitize-moduulin $sanitize-palvelulla, jottei epäluotettavaa dataa pääsisi palvelinpuolelle ollenkaan [79]. XSS tulee oleellisemmaksi huoleksi mahdollisen sovellukseen tulevan organisaatiorakennemuutoksen yhteydessä, jonka on tarkoitus mahdollistaa käyttäjien välinen projektien jakaminen.

4. Epäluotettavat suorat olioviittaukset

Epäluotettavilla suorilla olioviittauksilla hyökkääjä voi päästä käsiksi sovelluksen sisäisiin tietoihin, kuten tietokantadataan, tiedostoihin ja kansioihin, joihin hänellä ei pitäisi olla pääsyä. Tähän on usein syynä huono sovelluksen sisäinen pääsynvalvonta. [75.]

Sovelluksen istunnot ja tietokantayhteydet on toteutettu Laravelin toteutuksia käyttäen niin, että jokaisella käyttäjällä on pääsy vain omiin tietoihinsa. Käyttäjien pääsyä valvotaan jokaisen tarpeellisen resurssin kohdalla tarkistamalla, että käyttäjä on kirjautunut sovellukseen. Osoitepolkuihin ei ole myöskään mahdollista lisätä tietoja, jotka voisivat päätyä olioviittauksiin.

5. Virheelliset turvallisuusasetukset

Virheelliset turvallisuusasetukset ja sovelluksen käyttämien riippuvuuksien vanhentuneet versiot luovat haavoittuvaisuuksia [75].

Sovelluksen riippuvuuksien oletusarvoisia asetuksia on muutettava. Etenkin oletusarvoiset salasanat tulisi vaihtaa ja vain sovelluskehitykseen tarkoitetut ominaisuudet on otettava pois käytöstä ennen sovelluksen käyttöönottoa. Lisäksi riippuvuudet on pidettävä päivitettyinä ja kansioiden ja tiedostojen käyttöoikeuksien on oltava asetettu niin, etteivät ulkopuoliset pääse tekemään niihin muutoksia.

6. Arkaluonteisen datan altistuminen käyttäjille

Arkaluonteisen datan altistuminen käyttäjille saattaa mahdollistaa esimerkiksi identiteettivarkaudet sovelluksessa [75].

Käyttäjien salasanat ovat laskentasovelluksen arkaluonteisimmat tiedot ja ne salataan Laravelin Hash-luokan käyttämän salausalgoritmin avulla ennen tallentamista tietokantaan [77]. Sovelluksen käyttöönoton yhteydessä on vielä otettava käyttöön salausprotokollaa käyttävä yhteys.

7. Pääsy käyttäjiltä estetyksi tarkoitettuun toiminnallisuuteen

Pääsy käyttäjiltä estetyksi tarkoitettuun toiminnallisuuteen voi tahattomasti antaa käyttäjille mahdollisuuden käyttää sovellusta ilman kirjautumista tai pahimmassa tapauksessa pääsyn ylläpito-oikeuksia vaativille sivuille [75].

Käyttäjien pääsyä valvotaan jokaisen tarpeellisen resurssin kohdalla tarkistamalla, että käyttäjä on kirjautunut sovellukseen. Pääsy tulevaisuudessa mahdollisesti kehitettäviin järjestelmän ylläpitäjän ylläpitosivuihin tulee estää normaalikäyttäjiltä.

8. Cross-Site Request Forgery (CSRF)

CSRF-hyökkäys pakottaa sovellukseen kirjautuneen käyttäjän selainta lähettämään palvelimelle HTTP-pyynnön, jonka palvelin luulee tulevan omalta kirjautuneelta käyttäjältään. Pyyntö voidaan lähettää käyttäjän huomaamatta käyttäjän vieraillessa ulkoisessa ja haitallisessa sivustossa. Hyökkääjä pääsee tällä tavalla käsiksi muuten hänellä ulottumattomissa oleviin toiminnallisuuksiin, kuten käyttäjätunnuksen tietojen muokkaamiseen. [75.]

Laravel luo sovelluksen latauksen yhteydessä CSRF-tunnuksen, joka tallennetaan asiakaspuolelle. Tunnus otetaan mukaan tarkistettavaksi tiettyihin sovelluksessa tehtäviin HTTP-pyyntöihin. Ulkoisilla sivuilla ei ole pääsyä tunnukseen. [77.]

9. Haavoittuvaisuuksia sisältävien komponenttien käyttö

Sovelluksen käyttämät ulkoiset komponentit toimivat lähes aina täysillä oikeuksilla. Haavoittuvaisuuksia sisältävien komponenttien käyttö voi näin ollen avata useitakin jo aiemmin mainittuja uhkia ja saada sovelluksessa merkittävää vahinkoa aikaiseksi. [75.]

Sovelluksen kehittämiseen on käytetty useita avoimeen lähdekoodiin perustuvia ja useimmiten suurten kehittäjäyhteisöjen ylläpitämiä komponentteja. Näiden komponenttien, etenkin tietoturvallisuuteen liittyviä päivityksiä tulisi seurata. Ulkoisten komponenttien sisältöihin on myös osattava suhtautua kriittisesti.

10. Tahaton sivulta ohjaaminen vahvistamattomille sivuille

Verkkosovellukset ohjaavat käyttäjiä usein muille sivuille ja käyttävät epäluotettavaa dataa päämääräsivujen määrittämisessä. Jos uudelleenohjaukset on toteutettu huolimattomasti, voivat hyökkääjät ohjata käyttäjiä huijaus- ja haittaohjelmasivuille käyttäen osoitteessa luotettavan sivun nimeä ja jotakin toiselle sivulle ohjaavaa tietoa. [75.]

Sovellukseen on tehty tarkat säännöt sovelluksen sallimia osoitepolkuja varten. Säännöt on määritetty sovelluksen käyttämien sovelluskehysten avulla. Hyökkääjien ei ole siis mahdollista luoda osoitteita, jotka ohjaisivat käyttäjiä laskentasovelluksen sisäisen luotettavan näköisen osoitteen kautta haitallisille sivuille.

Ennen laskentasovelluksen käyttöönottoa tulisi vielä ottaa useita asioita huomioon. Sovellukseen täytyy ottaa käyttöön salausprotokollaa käyttävä yhteys, kaikki sovelluksen kansioiden ja tiedostojen käyttöoikeudet on konfiguroitava oikein ja eri riippuvuuksien oletusasetuksia on muutettava etenkin salasanojen osalta. Myös kaikki pelkästään kehitysympäristöön tarkoitetut ominaisuudet on otettava pois käytöstä. Lisäksi käyttäjien syötteet täytyy puhdistaa ennen palvelimelle lähettämistä ja käyttäjien todentaminen on otettava vielä tarkempaan tarkasteluun. OWASPin listaan on sisällytetty vain kymmenen merkittävintä tietoturvauhkaa, joten muistakin tietoturvallisuuteen liittyvistä asioista on vielä syytä ottaa selvää.

6 Yhteenveto

Tietotekniikan ala on koko ajan liikkumassa pois päin vanhasta PC-keskeisestä ajattelutavasta mobiilimmaksi ja uusiin alustoihin, kuten pilvipalveluihin. Tietotekniikkaan, tietoliikennetekniikkaan ja kuluttajateknologioihin erikoistuneen tutkimuslaitoksen International Data Corporationin ennusteen mukaan vuonna 2015 pilvipalveluiden ekosysteemiin tullaan kuluttamaan yhä suurempia rahamääriä. Kansainvälisen tietotekniikan tutkimuslaitos Gartnerin tekemän tutkimuksen mukaan yritysten usko pilvipalvelujen kykyyn toimittaa edullisempia, innovatiivisempia ja joustavampia ratkaisuja yrityskäyttöön on suurempi kuin koskaan. Sovelluksia ja palveluja tulee siis todennäköisesti jatkossa tarjolle yhä pienemmille asiakaskunnille ja yksityiskohtaisempiin käyttötarkoituksiin, kuten tämänkin insinöörityönä kehitetyn laskentasovelluksen tapauksessa voidaan osittain todeta. [80; 81.]

Tavoitteena oli insinöörityötä aloittaessa saada Excel-työkirjaan sisäänrakennettu laskentatyökalu toteutettua verkkosovelluksena kaikkine työkirjatiedostotyökalun ominaisuuksineen. Laskentalogiikan lisäksi verkkosovellukseen piti saada toteutettua käyttäjärakenteet, raportointiominaisuus ja sovelluksen tuli toimia projektipohjaisena. Tallennusmenetelmien toteuttaminen ja visuaalisesta taulukkomaailmasta siirtyminen vähemmän visuaaliseen ohjelmointimaailmaan toivat työhön omat haasteensa.

Työ alkoi rajaamalla työn tavoitteet ja asettamalla toteutettavat ominaisuudet tärkeysjärjestykseen. Sovelluksessa oli valtavasti logiikkaa ja vaikka laskentalogiikka ei itsessään muodostunutkaan erityisen hankalaksi osa-alueeksi, sisälsi se silti paljon riippuvuuksia toisiin tietoihin ja muita aikaa vieviä erikoisuuksia. Tämän vuoksi jo laskentalo-

giikan toteuttamiseen kului paljon aikaa. Työtä päädyttiin siis rajaamaan niin, ettei sovelluksen ulkoasuun liittyviin ominaisuuksiin kiinnitettäisi insinöörityön osalta erityisemmin huomiota. Koimme insinöörityön tilaajan kanssa tärkeimmäksi sen, että sovelluksen toiminta saadaan kuitenkin kokonaisuudessaan toteutettua ja tämän otimmekin työn osalta ensisijaiseksi tavoitteeksi.

Yhteistyöni tilaajan kanssa sujui projektin aikana kaiken kaikkiaan hyvin ja työn tekeminen oli mukavaa. Varsinainen kehitystyö toteutettiin etätyönä tilaajan tarjoamalla palvelinalustalla. Tilaaja koosti laskentalogiikasta pseudokoodia eli englannin kielen kaltaista kieltä, jonka tarkoituksena on kuvata toteutettavia algoritmeja lyhyesti ja ytimekkäästi riippumatta ohjelmointikielten välisistä syntaksieroista. Tilaajan koostamista pseudokoodeista oli korvaamaton apu sovelluksen laskentalogiikan ohjelmoinnissa. Sähköposti ja puhelin olivat täysin toimivia välineitä projektiviestinnässä. Lisäksi pidimme tilaajan kanssa tarvittaessa pidempiä etäpalavereja internetpuheluiden avulla.

Työlle asetetut tavoitteet saatiin toteutettavan prototyypin osalta toteutettua. Sovelluksen käyttäjät pystyvät kirjautumaan sovellukseen ja työstämään sovelluksen sisällä projekteja. Projekteja pystytään tallentamaan palvelimen tietokantaan ja omalle työasemalle. Varsinaista arvoa sovellus tuottaa käyttäjilleen erilaisilla raporteilla, joita sovellus luo käyttäjien syöttämien syöttötietojen perusteella. Myös raportointiominaisuus saatiin toteutettua kaikilla toivotuilla tavoilla, joita olivat raporttien tuottaminen CSV- ja PDF-tiedostoina sekä raportin näyttäminen suoraan käyttäjän selaimessa HTML-muodossa.

Täydellistä lopputulosta ei kuitenkaan saatu aikaiseksi. Laskentaa voidaan suorittaa sovelluksessa lukuisilla eri syöttötiedoilla eikä testauksen jäädessä vähäiseksi voida olla vielä täysin varmoja laskennan virheettömästä toiminnasta. Sovellukseen tuli myös projektin aikana useita muutoksia, joiden vuoksi joitakin jo valmiitakin ominaisuuksia jouduttiin muokkaamaan. Muutosten tekemiseen kului myös odottamattomasti aikaa. Sovelluksen tietoturvallisuusasioihin ja käyttöliittymän ulkoasuun joudutaan myös kiinnittämään jatkossa erityistä huomiota. Tilaaja oli kuitenkin yleisesti ottaen tyytyväinen työpanokseeni ja ymmärtää itsekin työn laajuuden.

Lähteet

1 Heino, Petteri. 2010. Pilvipalvelut. Hämeenlinna: Talentum Media Oy.

2 Encoding PHP Files to Protect your PHP Application. 2015. Verkkodokumentti. PHP-Editors. <http://www.php-editors.com/php-tools/php-encoder.php>. Luettu 27.3.2015.

3 Leskinen, JR. 2015. Mobiili laajakaista vihdoin koko maahan. Verkkodokumentti. Kauppalehti. <http://www.kauppalehti.fi/uutiset/mobiili-laajakaista-vihdoin-koko-maahan/DN8BGZCx>. Päivitetty 28.1.2015. Luettu 27.3.2015.

4 Salo, Immo. 2012. Hyötyä pilvipalveluista. Jyväskylä: Docendo Oy.

5 Cloud Computing Outlook: IaaS, PaaS and SaaS. 2015. Verkkodokumentti. QArea. <http://www.qarea.com/articles/cloud-computing-outlook-iaas-paas-and-saas>. Luettu 28.3.2015.

6 What is SaaS? 2015. Verkkodokumentti. Interoute Communications Limited. <http://www.interoute.com/what-saas>. Luettu 28.3.2015.

7 Miller, Michael. 2009. Cloud Computing: Web-Based Applications That Change the Way You Work and Collaborate Online. Indianapolis: Que Publishing.

8 Kohan, Bernard. 2015. Guide to Web Application Development. Verkkodokumentti. Comentum Corp. <http://www.comentum.com/guide-to-web-application-development.html>. Luettu 28.3.2015.

9 Jacobs, Ian & Walsh, Norman. 2004. Architecture of the World Wide Web, Volume One. Verkkodokumentti. The World Wide Web Consortium. <http://www.w3.org/TR/webarch/>. Päivitetty 15.12.2004. Luettu 28.3.2015.

10 Fielding, Roy T. & Reschke, Julian F. 2014. Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. Verkkodokumentti. Internet Engineering Task Force (IETF). <http://tools.ietf.org/html/rfc7230>. 2014. Luettu 28.3.2015.

11 PHP: Hypertext Preprocessor. 2015. Verkkodokumentti. PHP. <http://php.net/>. Luettu 28.3.2015.

12 Usage of server-side programming languages for websites. 2015. Verkkodokumentti. W3Techs. <http://w3techs.com/technologies/overview/programming_language/all>. Luettu 29.3.2015.

13 Rouse, Margaret. 2008. LAMP (Linux, Apache, MySQL, PHP). Verkkodokumentti. TechTarget. <http://searchenterpriselinux.techtarget.com/definition/LAMP>. 2008. Luettu 29.3.2015.

14 HTML developer guide. 2015. Verkkodokumentti. Mozilla Developer Network. <https://developer.mozilla.org/en-US/docs/Web/Guide/HTML>. Päivitetty 14.1.2015. Luettu 29.3.2015.

15 CSS. 2015. Verkkodokumentti. Mozilla Developer Network. <https://developer.mozilla.org/en-US/docs/Web/CSS>. Päivitetty 15.2.2015. Luettu 29.3.2015.

16 JavaScript. 2015. Verkkodokumentti. Mozilla Developer Network. <https://developer.mozilla.org/en-US/docs/Web/JavaScript>. Päivitetty 24.3.2015. Luettu 29.3.2015.

17 DOM. 2015. Verkkodokumentti. Mozilla Developer Network. <https://developer.mozilla.org/en-US/docs/Glossary/DOM>. Päivitetty 28.3.2015. Luettu 29.3.2015.

18 Garret, Jesse James. 2005. Ajax: A New Approach to Web Applications. Verkkodokumentti. Adaptive Path. <http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications/>. Päivitetty 18.2.2005. Luettu 29.3.2015.

19 Pierce, Benjamin C. 2002. Types and Programming Languages. Massachusetts: The MIT Press.

20 Hunt, Andrew & Thomas, David. 2010. The Pragmatic Programmer: From Journeyman to Master. Massachusetts: Addison Wesley Longman, Inc.

21 Hayder, Hasin. 2007. Object-Oriented Programming with PHP5. Birmingham: Packt Publishing Ltd.

22 CSV File Format. 2015. Verkkodokumentti. CSV Reader. <http://www.csvreader.com/csv_format.php>. Luettu 30.3.2015.

23 About Adobe PDF. 2015. Verkkodokumentti. Adobe Systems Incorporated. <http://www.adobe.com/products/acrobat/adobepdf.html>. Luettu 30.3.2015.

24 Wasson, Mike. 2013. Single-Page Applications: Build Modern, Responsive Web Apps with ASP.NET. Verkkodokumentti. MSDN Magazine. <https://msdn.microsoft.com/en-us/magazine/dn463786.aspx>. 2013. Luettu 30.3.2015.

25 Heiskanen, Henri. 2013. Yhden sivun web-sovellukset tulevat, oletko valmis? Verkkodokumentti. Gofore. <https://gofore.com/ohjelmistokehitys/yhden-sivun-web-sovellukset-tulevat-oletko-valmis/>. Päivitetty 31.5.2013. Luettu 30.3.2015.

26 Introducing JSON. 2015. Verkkodokumentti. JSON. <http://json.org/>. Luettu 30.3.2015.

27 The PHP Framework For Web Artisans. 2015. Verkkodokumentti. Laravel. <http://laravel.com/>. Luettu 30.3.2015.

28 Architecture of Laravel Applications. 2015. Verkkodokumentti. Laravel Book. <http://laravelbook.com/laravel-architecture/>. Luettu 30.3.2015.

29 Facades. 2015. Verkkodokumentti. Laravel. <http://laravel.com/docs/4.2/facades>. Luettu 31.3.2015.

30 IoC Container. 2015. Verkkodokumentti. Laravel. <http://laravel.com/docs/4.2/ioc>. Luettu 31.3.2015.

31 Routing. 2015. Verkkodokumentti. Laravel. <http://laravel.com/docs/4.2/routing>. Luettu 31.3.2015.

32 Mehta, Nisarg J. 2014. Architecture of API (Application Programming Interface). Verkkodokumentti. Techtic Solutions. <https://www.techtic.com/blog/architecture-of-api-application-programming-interface/>. Päivitetty 16.7.2014. Luettu 31.3.2015.

33 Controllers. 2015. Verkkodokumentti. Laravel. <http://laravel.com/docs/4.2/controllers>. Luettu 31.3.2015.

34 Vaqqas, M. 2014. RESTful Web Services: A Tutorial. 2015. Verkkodokumentti. Dr. Dobb's. <http://www.drdobbs.com/web-development/restful-web-services-a-tutorial/240169069>. Päivitetty 23.9.2014. Luettu 1.4.2015.

35 Errors & Logging. 2015. Verkkodokumentti. Laravel. <http://laravel.com/docs/4.2/errors>. Luettu 1.4.2015.

36 Views & Responses. 2015. Verkkodokumentti. Laravel. <http://laravel.com/docs/4.2/responses>. Luettu 1.4.2015.

37 Laravel Repository Pattern. 2015. Verkkodokumentti. Vegibit. <http://vegibit.com/laravel-repository-pattern/>. Luettu 1.4.2015.

38 Requests & Input. 2015. Verkkodokumentti. Laravel. <http://laravel.com/docs/4.2/requests>. Luettu 1.4.2015.

39 AngularJS — Superheroic JavaScript MVW Framework. 2015. Verkkodokumentti. AngularJS. <https://angularjs.org/>. Luettu 2.4.2015.

40 AngularJS: Developer Guide: Data Binding. 2015. Verkkodokumentti. AngularJS. <https://docs.angularjs.org/guide/databinding>. Luettu 2.4.2015.

41 AngularJS: Developer Guide: Modules. 2015. Verkkodokumentti. AngularJS. <https://docs.angularjs.org/guide/module>. Luettu 2.4.2015.

42 AngularJS: API: ngRoute. 2015. Verkkodokumentti. AngularJS. <https://docs.angularjs.org/api/ngRoute>. Luettu 2.4.2015.

43 AngularJS: API: $routeProvider. 2015. Verkkodokumentti. AngularJS. <https://docs.angularjs.org/api/ngRoute/provider/$routeProvider>. Luettu 3.4.2015.

44 AngularJS: API: $http. 2015. Verkkodokumentti. AngularJS. <https://docs.angularjs.org/api/ng/service/$http>. Luettu 3.4.2015.

45 AngularJS: Developer Guide: Providers. 2015. Verkkodokumentti. AngularJS. <https://docs.angularjs.org/guide/providers>. Luettu 3.4.2015.

46 AngularJS: Developer Guide: Controllers. 2015. Verkkodokumentti. AngularJS. <https://docs.angularjs.org/guide/controller>. Luettu 3.4.2015.

47 AngularJS: Developer Guide: Scopes. 2015. Verkkodokumentti. AngularJS. <https://docs.angularjs.org/guide/scope>. Luettu 3.4.2015.

48 AngularJS: Developer Guide: Filters. 2015. Verkkodokumentti. AngularJS. <https://docs.angularjs.org/guide/filter>. Luettu 3.4.2015.

49 AngularJS: API: ngSubmit. 2015. Verkkodokumentti. AngularJS. <https://docs.angularjs.org/api/ng/directive/ngSubmit>. Luettu 3.4.2015.

50 AngularJS: Developer Guide: Directives. 2015. Verkkodokumentti. AngularJS. <https://docs.angularjs.org/guide/directive>. Luettu 3.4.2015.

51 AngularJS: API: ngModel. 2015. Verkkodokumentti. AngularJS. <https://docs.angularjs.org/api/ng/directive/ngModel>. Luettu 3.4.2015.

52 AngularJS: API: ngRepeat. 2015. Verkkodokumentti. AngularJS. <https://docs.angularjs.org/api/ng/directive/ngRepeat>. Luettu 3.4.2015.

53 AngularJS: API: ngClick. 2015. Verkkodokumentti. AngularJS. <https://docs.angularjs.org/api/ng/directive/ngClick>. Luettu 3.4.2015.

54 AngularJS: API: ngApp. 2015. Verkkodokumentti. AngularJS. <https://docs.angularjs.org/api/ng/directive/ngApp>. Luettu 3.4.2015.

55 AngularJS: API: ngView. 2015. Verkkodokumentti. AngularJS. <https://docs.angularjs.org/api/ngRoute/directive/ngView>. Luettu 3.4.2015.

56 Basic Database Usage. 2015. Verkkodokumentti. Laravel. <http://laravel.com/docs/4.2/database>. Luettu 4.4.2015.

57 Eloquent ORM. 2015. Verkkodokumentti. Laravel. <http://laravel.com/docs/4.2/eloquent>. Luettu 4.4.2015.

58 Session. 2015. Verkkodokumentti. Laravel. <http://laravel.com/docs/4.2/session>. Luettu 4.4.2015.

59 JSON: The Fat-Free Alternative to XML. 2015. Verkkodokumentti. JSON. <http://json.org/xml.html>. Luettu 4.4.2015.

60 JSON Schema and Hyper-Schema. 2015. Verkkodokumentti. JSON Schema. <http://json-schema.org/>. Luettu 4.4.2015.

61 Galiegue, F., Zyp, K. & Court, G. JSON Schema: core definitions and terminology. 2013. Verkkodokumentti. JSON Schema. <http://json-schema.org/latest/json-schema-core.html>. Päivitetty 30.1.2013. Luettu 4.4.2015.

62 Galiegue, F., Zyp, K. & Court, G. JSON Schema: interactive and non interactive validation. 2013. Verkkodokumentti. JSON Schema. <http://json-schema.org/latest/json-schema-validation.html>. Päivitetty 30.1.2013. Luettu 4.4.2015.

63 Templates. 2015. Verkkodokumentti. Laravel. <http://laravel.com/docs/4.2/templates>. Luettu 5.4.2015.

64 DOMPDF Wrapper for Laravel 4. 2015. Verkkodokumentti. GitHub, Inc. <https://github.com/barryvdh/laravel-dompdf/tree/0.4>. Luettu 5.4.2015.

65 Snappy PDF/Image Wrapper for Laravel 4. 2015. Verkkodokumentti. GitHub, Inc. <https://github.com/barryvdh/laravel-snappy/tree/0.1>. Luettu 5.4.2015.

66 Manifesto for Agile Software Development. 2015. Verkkodokumentti. The Agile Manifesto. <http://agilemanifesto.org/>. Luettu 5.4.2015.

67 phpDocumentor. 2015. Verkkodokumentti. phpDocumentor. <http://www.phpdoc.org/>. Luettu 5.4.2015.

68 Angular-JSDoc. 2015. Verkkodokumentti. GitHub, Inc. <https://github.com/allenhwkim/angular-jsdoc>. Luettu 5.4.2015.

69 Laravel Cashier. 2015. Verkkodokumentti. Laravel. <http://laravel.com/docs/4.2/billing>. Luettu 6.4.2015.

70 Stripe: Pricing. 2015. Verkkodokumentti. Stripe. <https://stripe.com/fi/pricing>. Luettu 6.4.2015.

71 PHPUnit – The PHP Testing Framework. 2015. Verkkodokumentti. PHPUnit. <https://phpunit.de/>. Luettu 6.4.2015.

72 Mockery Docs. 2015. Verkkodokumentti. Mockery. <http://docs.mockery.io/en/latest/>. Luettu 6.4.2015.

73 AngularJS: Developer Guide: Unit Testing. 2015. Verkkodokumentti. AngularJS. <https://docs.angularjs.org/guide/unit-testing>. Luettu 6.4.2015.

74 AngularJS: Developer Guide: E2E Testing. 2015. Verkkodokumentti. AngularJS. <https://docs.angularjs.org/guide/e2e-testing>. Luettu 6.4.2015.

75 OWASP Top 10 - 2013. 2013. Verkkodokumentti. The OWASP Foundation. <http://owasptop10.googlecode.com/files/OWASP%20Top%2010%20-%202013.pdf>. 2013. Luettu 7.4.2015.

76 Query Builder. 2015. Verkkodokumentti. Laravel. <http://laravel.com/docs/4.2/queries>. Luettu 7.4.2015.

77 Security. 2015. Verkkodokumentti. Laravel. <http://laravel.com/docs/4.2/security>. Luettu 7.4.2015.

78 AngularJS: Miscellaneous: FAQ. 2015. Verkkodokumentti. AngularJS. <https://docs.angularjs.org/misc/faq>. Luettu 7.4.2015.

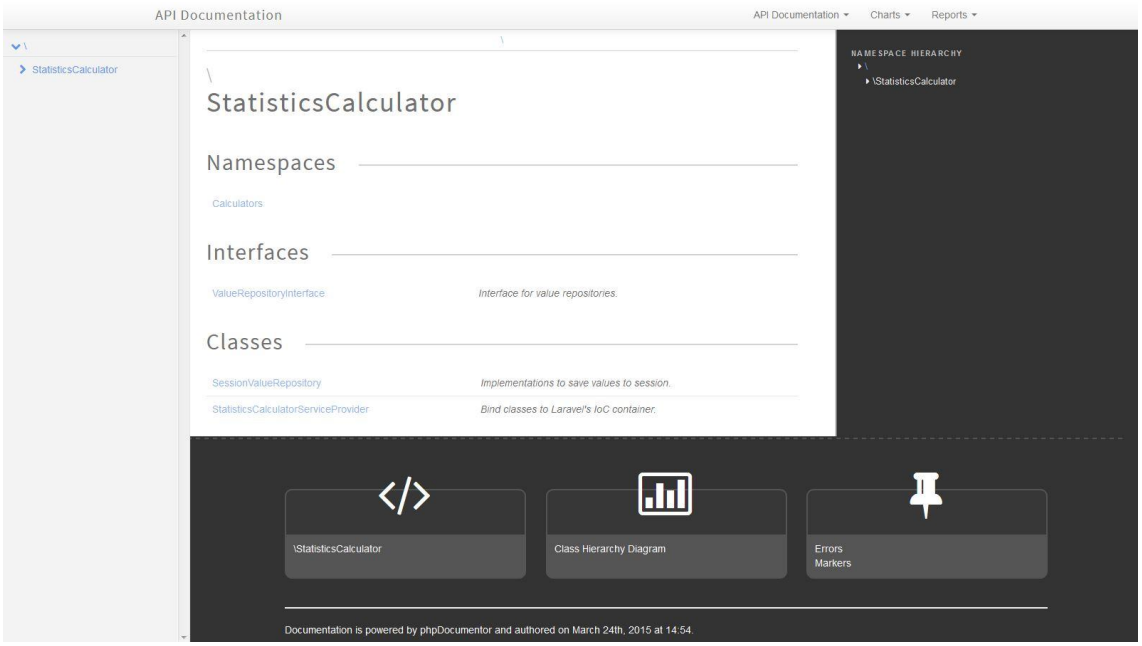
79 AngularJS: API: $sanitize. 2015. Verkkodokumentti. AngularJS. <https://docs.angularjs.org/api/ngSanitize/service/$sanitize>. Luettu 7.4.2015.

80 IDC Predicts the 3rd Platform Will Bring Innovation, Growth, and Disruption Across All Industries in 2015. 2014. Verkkodokumentti. International Data Corporation (IDC). <http://www.idc.com/getdoc.jsp?containerId=prUS25285614>. Päivitetty 2.12.2014. Luettu 7.4.2015.

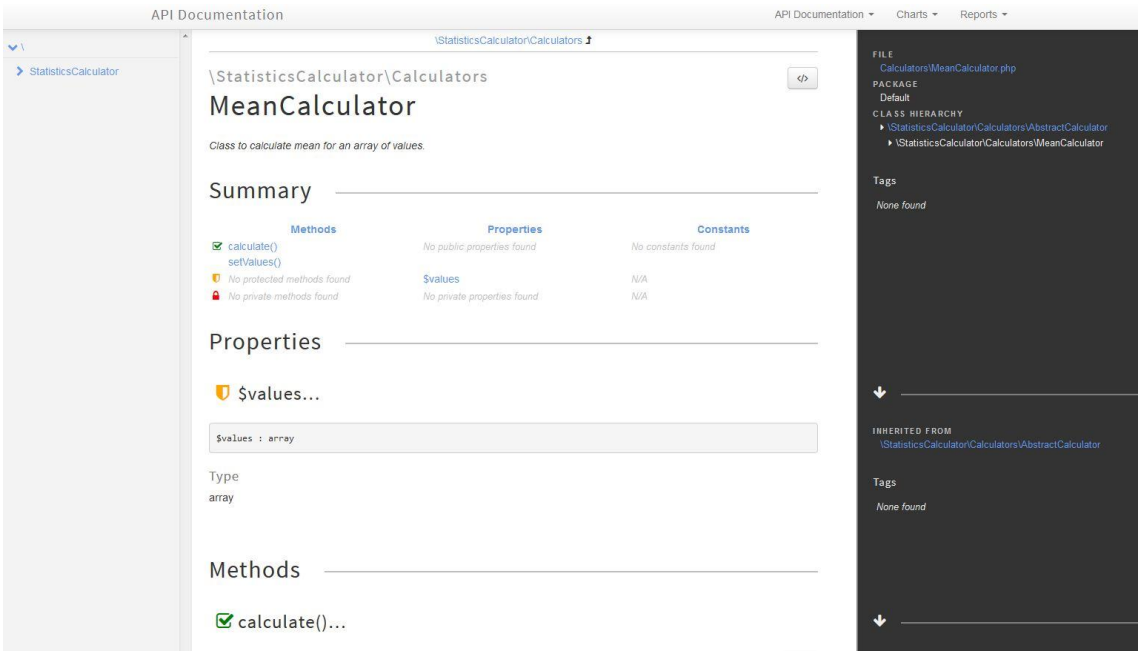
81 Gartner Survey Reveals That SaaS Deployments Are Now Mission Critical. 2014. Verkkodokumentti. Gartner, Inc. <http://www.gartner.com/newsroom/id/2923217>. Päivitetty 25.11.2014. Luettu 7.4.2015.

Esimerkki phpDocumentor-työkalun luomasta dokumentoinnista

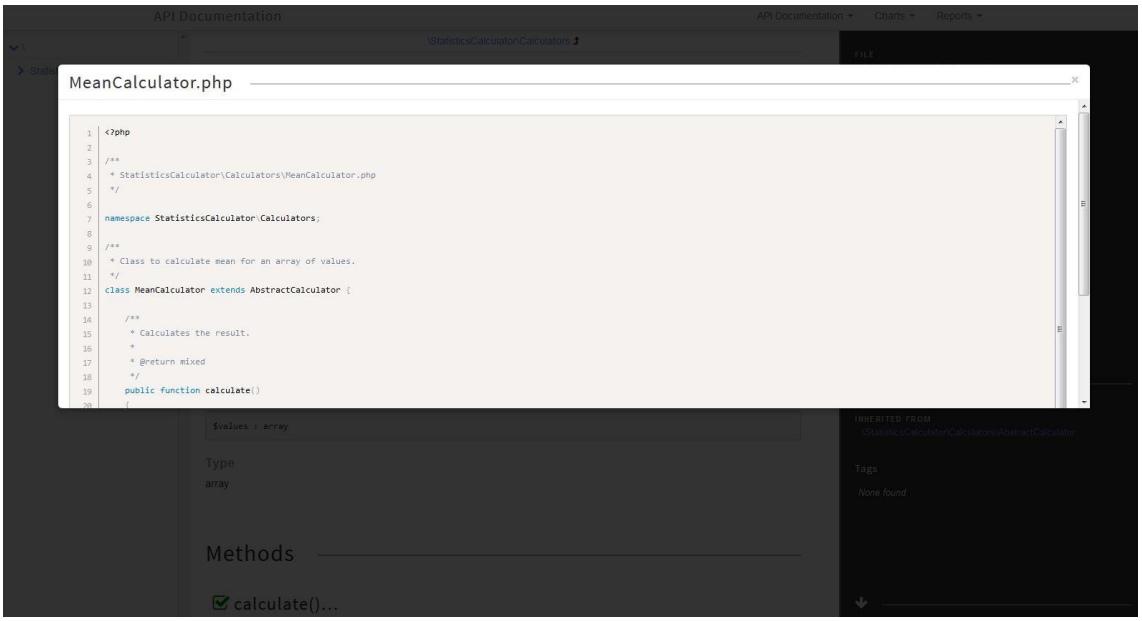
Dokumentoinnin yleisnäkymä.



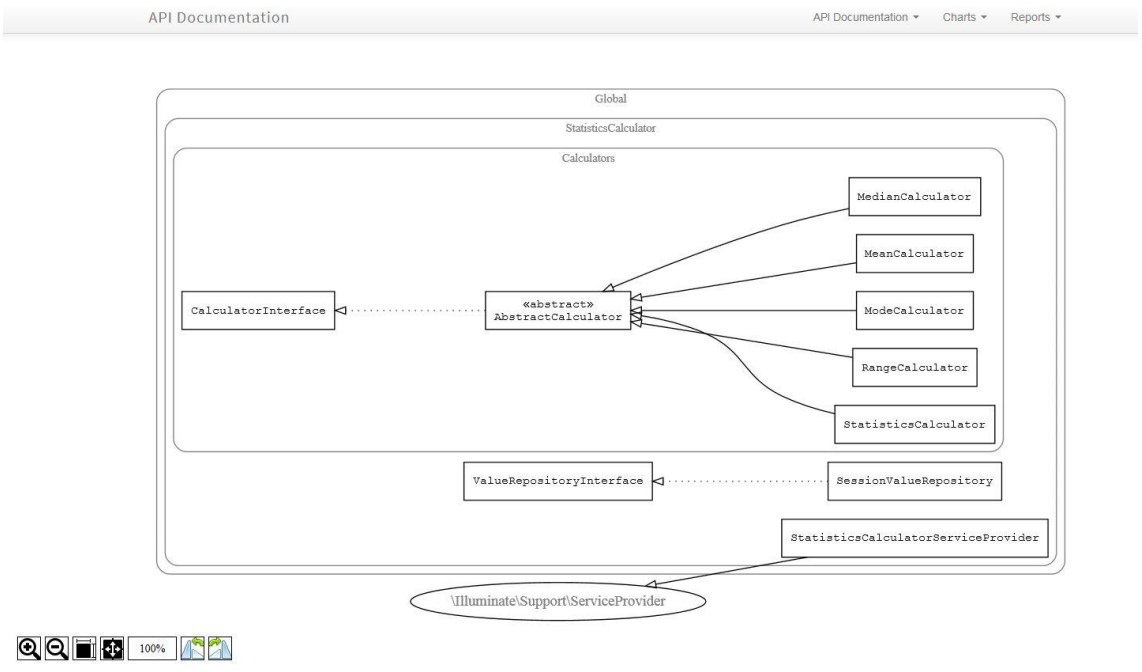
Yksittäisen luokan tarkastelunäkymä.



Lähdekoodin lukeminen suoraan selaimella.

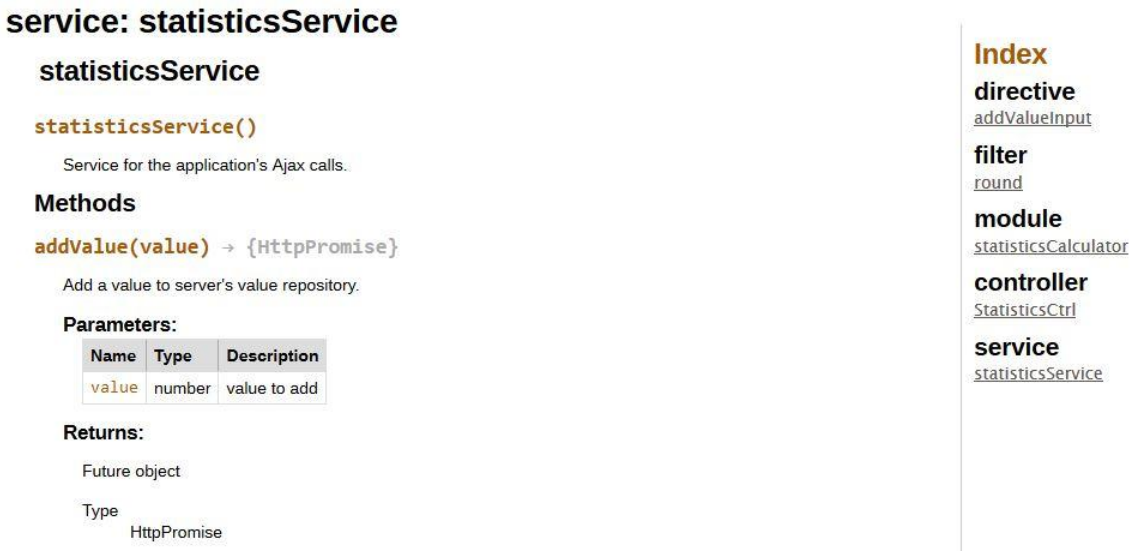


Työkalun luoma luokkakaavio.



Esimerkki Angular-JSDoc-työkalun luomasta dokumentoinnista

Näkymä yksittäisen AngularJS-komponentin dokumentoinnista.



service: statisticsService

statisticsService

statisticsService()

Service for the application's Ajax calls.

Methods

addValue(value) → {HttpPromise}

Add a value to server's value repository.

Parameters:

Name	Type	Description
value	number	value to add

Returns:

Future object

Type
HttpPromise

Index
directive
addValueInput
filter
round
module
statisticsCalculator
controller
StatisticsCtrl
service
statisticsService

Näkymä yksittäisen AngularJS-komponentin lähdekoodista.

Source: services/statisticsService.js

```
1.  /**
2.   * @ngdoc service
3.   * @name statisticsService
4.   * @description
5.   * Service for the application's Ajax calls.
6.   */
7.  app.factory('statisticsService', function($http) {
8.
9.      return {
10.
11.         /**
12.          * Get saved values and calculated statistics.
13.          *
14.          * @memberof statisticsService
15.          *
16.          * @returns {HttpPromise} Future object
17.          */
18.         getData: function() {
19.             return $http.get('/api/statistics');
20.         },
```

Index
directive
addValueInput
filter
round
module
statisticsCalculator
controller
StatisticsCtrl
service
statisticsService